IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of: Duvdevani et al. Docket No: U 014859-9 (46766/907)

Appln. No.: 10/706,489 Group Art Unit: 2623

Confirmation No.: 7213 Examiner: KIBLER, Virginia M.

Filed: 11/12/2003

For: APPARATUS AND METHOD FOR THE INSPECTION OF OBJECTS

Honorable Commissioner for Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450

DECLARATION UNDER 37 CFR 1.131

Sir:

I, the undersigned, Shmuel Rippa, hereby declare as follows:

1) I am an Applicant in the patent application identified above, and I am one of the co-inventors of the subject matter described and claimed in claims 1 – 16 therein.

2) In 1996 I joined a the ICP group of the Assignee as a software engineer on a software development team assigned to develop and implement a software system performing the functionalities described and claimed in the subject Application in systems for inspecting lead frames, and eventually ball grid array substrates, being developed at the time by the Assignee. I later advanced and became team leader of the software development team.

3) Prior to February 5, 1998, our team reduced to practice the invention described and claimed in the subject application, *inter alia*, by developing working software code that was adapted to inspect ball grid array substrates. Our team did its work in Israel, a WTO country.

4) Appendix A contains a paper that I wrote on or before October 29, 1996 entitled, "SIP Tests in Windows", and stored as a computer file under the name "sip_tests_in_windows.fm*". "SIP" is an acronym for "Software Image Processing" Appendix A is part of system

documentation for a generic model for a software defect detection package implementing the invention described and claimed in the subject application.

I have printed Appendix "A" without making any modification whatsoever to the document as stored on the Assignee's computer system. The date appearing in the header of Appendix A is automatically inserted by a date field to display the date upon which the document was printed. A screen shot of the computer directory in which Appendix A is stored is contained in Appendix B and indicates that Appendix A was last modified on October 29, 1996.

5) Appendix C contains a paper that I wrote and printed out on or before June 28, 1998. "ICP" is an acronym for "Integrated Circuit Packaging" and is the name of a project of Assignee for inspection equipment designed to inspect ball grid array substrates, a type of integrated circuit packaging. We designed ICP inspection equipment to employ software inspection as described and claimed in the subject application.

6) Appendix D contains working software code, written in the "C" programming language, implementing a task packer for assigning inspection tasks to portions of interest in an image of an electrical circuit to be inspected, as referenced in Appendices A and C.

Appendix E contains working software code, written in the "C" programming language, implementing a task manager for managing the execution of inspection tasks assigned to portions of interest by the Code in Appendix D. The software programs in Appendices D & E are part of, and run with, a suite of software programs performing defect detection during the inspection of ball grid array substrates. Other programs in the suite of programs include, without limitation, various configuration files and specific inspection tasks assigned by a task packer employing a task packer helper as in Appendix D to portions of interest for inspecting corresponding portions of an electrical circuit to be inspected.

As evidenced by the revision log appearing on page 30 et seq. of Appendix D, Version 1.1 of the code in Appendix D is dated February 18, 1997. According to our practice, only

working and reviewed versions of code were numbered. As further evidenced by the revision log, this code was regularly updated at least until February 20, 2001

As evidenced by the revision log appearing on page 15 et seq. of Appendix E, Version 1.1 of the code in Appendix E is dated October 29, 1996. According to our practice, only working and reviewed versions of code were numbered. As further evidenced by the revision log, this code was regularly updated at least until February 7, 2000.

6) The following table shows the correspondence between the elements of the system claims in the present patent application and elements of the material in appendices.

Claim 1	**Appendices**
1. An electrical circuitry inspection method comprising:	The entire document, evidenced with particularity at the illustration on page 7 of Appendix A, marked "A".
for each of a plurality of types of local characteristics, each type occurring at least once within electrical circuitry to be inspected, identifying at least one portion of interest within the electrical circuitry whereat said local characteristic is expected to occur;	The illustration on page 7 of Appendix A marked "A", the text at page 2 of Appendix A marked "B" and the text at page and the text at page 4 of Appendix A marked "C".
and inspecting an image of each portion of interest, using an inspection task selected in response to the type of local characteristic expected to occur in the portion of interest.	The text at pages 4 & 5 of Appendix A marked "D" in combination with the text at page 4 of Appendix A marked "C".
Claim 2	**Appendices**

2. A method according to claim 1, wherein said plurality of types of local characteristics includes at least one of the following types: a bonding pad; a ball structure; a target; a chip area.	The text at page 5 of Appendix C marked "E".
Claim 3	**Appendices**
3. A method according to claim 1, wherein said identifying of at least one portion of interest comprises identification of at least one spatial region within said electrical circuitry.	The illustration on page 7 of Appendix A, marked "A".
Claim 4	**Appendices**
4. A method according to claim 3, wherein said identification of at least one spatial region is at least partly based on a user input.	The text at page 4 of Appendix A marked "C".
Claim 5	**Appendices**
5. A method according to claim 3, wherein said identification of at least one spatial region is at least partly based on a computer generated input.	The text at page 4 of Appendix A marked "C".
Claim 6	**Appendices**
6. A method according to claim 4, wherein said	The text at page 4 of Appendix A marked "C".

identification of at least one spatial region is at least partly based on a computer generated input.	
Claim 7	**Appendices**
7. A method according to claim 1, and also comprising computer-assigning an inspection task to at least one individual portion of interest in response to the type of local characteristic expected to occur in the individual portion of interest.	The text at page 4 & 5 of Appendix A marked "C" and "D". Text at page 5 of Appendix C marked "F". Appendix D is software code of a task manager for assigning a type dependent inspection task to portions of interest.
Claim 8	**Appendices**
8. A method according to claim 1, and also comprising outputting at least one indication of defects responsive to said inspecting step.	Text at pages 4 & 5 of Appendix A marked "D".

5) Claims 9 - – 16 recite apparatus for inspecting electrical circuits with limitations similar to those of system claims 1-8. Based on the similarity of subject matter between the apparatus and method claims for inspecting electrical circuits, it can similarly be demonstrated that the invention recited in claims 9 –16 was conceived and reduced to practice prior to July 28, 1999.

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and conjecture are thought to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are

punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application of any patent issued thereon.

Shmuel Rippa, Citizen of Israel
4 Rimalt St.
Ramat Gan
Israel

Date: December 8, 2004

Appendix "A"

20 October 2004
/home/asher-a/sip_tests_in_windows.fm

Orbotech Ltd.

SIP
Tests in Windows

Shmuel Rippa

1. Requirements

"Interesting" things usualy happens at particular locations which occupies only a small portion of the panel's area. Such are the areas near errors detected by hardware (for example, line width violation report) or by software (for example, excess/missing reports), areas around morphological events (centres, open-ends, etc.) and areas around predetermined locations which we call top down events. The top down events are defined by processing of the CAM of the board or by a special learn scan.

B

In order to accelerate the computation and allow more complicated image understanding algorithms we try to restrict the computation to small rectangular regions (windows) on the panel rather than doing the computation on the entire panel. Data generated by hardware and software modules and that is inside the specified window is added to the data of the window containing the data (this procedure is called packing of the data into the window). Examples of data generated by hardware include:

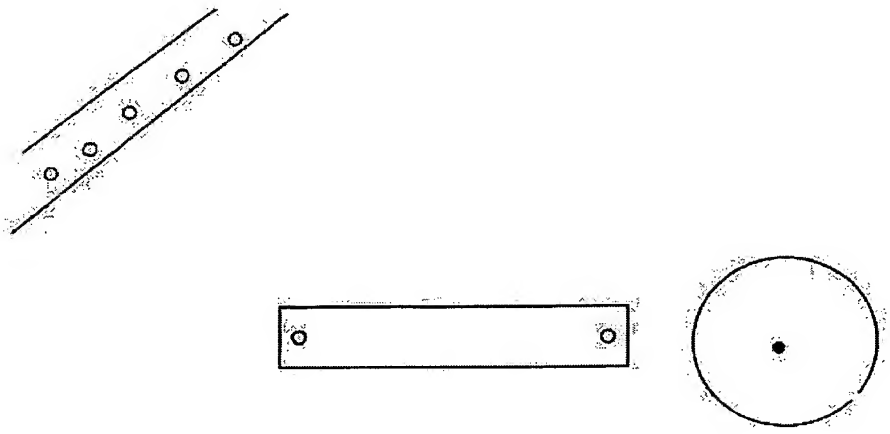
- Gray/Colour images of parts of the panel.
- CELs (Contour ELements) computed by special hardware cards.
- Morphology events (centers, open-ends, etc.) computed by hardware cards.
- More . . .

Examples of raw data genetrated by software modules include

- Excess/Missing reports
- Width violation reports.
- More . . .

Data entities are likely to be changed during the lifetime of a project as more data items are added and (perhaps) some items are removed from the above list.

In the following figure we present an example of a window containing a line segment (diagonal "fat" line), a circular shape (pad), and a rectangular shape (SMT pad) represented by CELs, two open-ends reports (displayed as dots at both ends of the SMT pad) and one center report (displayed as solid dot at the center of the circle). We also have various width violation reports (displayed as dots)on the line segment.



Data inside a window

Once the raw data is packed inside the window, it is ready to be processed by various algorithms. A typical algorithm operates on a window, uses the data inside the window and, perhaps, generates other data to be added to the data inside the window. In the following we give examples of some algorithms :

- Connected segment algorithm.
  The circle, rectangle and lines in the figure above are in fact represented by a collection of little line segments called CELs. These collections can be devided into four subsets. A CEL segment is in subset A iff it have a common point with another CEL from A. In the figure above we have four subsets : One for the circle, one for the rectangle and one for each line segment.
  Input : collection of CELS
  Output : subsets of conencted components ordered in a countercklockwise manner.
- Vectorizer
  Input : subset of connected components.
  Output : More compact representation of the subset (many little segments along a given line are replaced by a larger vector).
  Parameter: tolerance.
- Circle matcher. Match a circle in a least square sense to a set of points.
  Input : set of CELs
  Output: circle parameters.
- Shape finder. Searches for geometric enteties (circles, rectangles, lines, etc.). An algorithm should be written for each new geometric element that we wish to identify.
  Input : CELs or vectors from vectorizer.
  Output : Shape parameters.
  Parameters: Controling when the algorithm can safely declare on new shapoe.

The above algorithms can be used as basic blocks in higher level algorithms. For example an algorithm that searches for the presense of a circle will first devide all CELs into connected components. For each component the algorithm will decide either to vectorize the CELs or to leave them as is. In either case a call to circle shape finder will be made in order to check if the subset of conencted components contain a part of a circle. The algorithm will produce a list of

circles found. The algorithm also generates the subsets of connected components and the vector representation as byproducts. The total number of data items produced by higher level algorithm can not be determined in advance since it depends in the logic flow of the algorithm. There may be several algorithms for finding circles. For example if we knew that, in a specific panel, there are isolated circles and that each circle would be identified by a center report then we could device an efficient scheme that is given the set of CELs and the centre and finds the parameters of the circle around this centre.

Requirement 1 : To be able to present, graphically, each data item and all data items produced by a higher level algorithm.

Requirement 2 : To be able to select the algorithm to use, from a library of algorithms, from a configuration file and to supply parameters for the algorithm from that file. An example of such a parameter is a parameter that specifies how much CELs have to be sufficiently close to the circle that the algorithm found in order to decide that this is realy a circle.

Requirement 3 : To be able to add data items and algorithms without changing existing code.

Requirement 4 : Reusability of algorithms.

Result : We should be able to device a TCL/TK tool that displays the raw data and then allows us to select an algorithm and to specify the parameters for that algorithm. Then we apply the algorithm and can view all the data items produced by it.

nice to have : The ability to chain several simple and higher level algoriothms to form another algorithm. For example once we wrote an algorithm for finding circles and an algorithm for finding rectangles, we could create an algorithm for finding circles and rectangles. Ideally the algorithm should be implemented without the need for recompilation or relinking by using a scripting language (TCL). The scripting language should allow us to do some computations. For example checking that the centre of the circle is at a prescribed duistance from the centre of the rectangle. We want to be able to display, graphically, all data items produced by such a new algorithm.

The learning stage: In this stage we know very little about the scanned panel. We might have information that the panel is of a specific types (for example, it include only circles and rectangles). The data generated by hardware consists of CELs and features (centres and open-ends). We open a window around each feature. If the feature is a center, then we apply an algorithm that searches for a circle and if the feature is an open-end then we apply an algorithm that searches for a rectangle. The output of this stage is a reference file consisting of all circles and rectangles found in the panel. The reference may also include the original data CELs and features.

"C"

Requirement 5 : To be able to specify from a config file, the type of panel and thus the exact behaviour of the learning stage. We have to specify what algorithm to apply for each trigger and to specify what data items should be stored as reference.

The inspect stage: In this stage we are given the reference data created during the learn stage. For each ref data we attach a test function which will compare the raw data in the against the reference and decide if there is an error. The test function will be given all the

"D"

parameters required for making the comparison. For example a test for a ref data consisting of a circle might consist of computing the circle from the incoming data (CELs and features) and notify on error if the radius of the computed circle is different by more than a prescribed tolerance from the radius of the circle in the reference circle. Treatment of hardware/software errors is similar to the treatment in the learn stage.

"D"

Requirement 6: To be able to attach, from a config file, a test function for each item in the ref file and to be able to specify the parameters of the test function from the config file.

Requirement 7: To be able to view reference data agains raw data and to apply algorithms from TCL/TK tool.

2. Sipdata (sipdata.C, sipdata.H)

This is the base class for all data items used in window related algorithms. The class uses a factory (or virtual constructor) pattern, see also documentation in file sip_factory.H, so that derived classes can be added without need to change existing code. Each object derived from Sipdata have a type which is the name of the derived class and a name which uniquely identifies the object among many objects of the same type. The function `SetName` allows to set a name of a `Sipdata` object while the function `Name` returns that name. The base class Sipdata do not have a default constructor. It is necessary to specify a name for the object when constructing it. The Sipdata base class provides two more public methods, `Read` and `Write` for reading and writing of `Sipdata` objects. The `Write` method use `Io_util` object to write data in ASCII or binary forms. It writes the name of the derived class and then call `DoWrite` to let derived class do the rest of the job.Reads object from file in ascii or binary mode. The `Read` method function will be called after the calling program have read the line in the file containing the name of the derived class and called the `DoCreate` function to create an instance of the derived object. That `Read` method of that object is then invoked. Calls virtual `DoRead` function to do the actual reading.

Deriving from Sipdata.

In order to derive from Sipdata it is necessary to defined the following virtual functions:

- `Index`.
  The factory pattern requires that the base class will manage a table with an entry for each derived class. Index returns the index of the derived class of a specific object in that table.
- Type.
  Returns the name of the derived class.
- `DoCreate`.
  Allocate a new object from derived class and returns a (base class_ pointer to it
- `Display`
  Use a Display_data object to display graphically the concrete data of derived class.
- `Erase`
  Clears all data and leaves an empty object.
- `DoWrite`. Used by public method `Write` of base class.
  Writes object using `Io_util` object in ascii or binary mode. W
- `DoRead`
  Reads object written by `DoWrite` from file in ascii or binary mode.

The base class was designed so that it is possible to add new derived class in a very simple way

of 11

and without changing the code that is responsible for creation of derived objects stored in a file. To derive a class one have to folllow the instructions in `sipdata.H` file. Then all that is needed is to link the new code with the application.

Subclasses that have been derived so far are

Table 1: Subclassed derived from Sipdata

File names	Description
sipdata_cel_events.C sipdata_cel_events.H	collection of CELs
sipdata_feature_events.C sipdata_feature_events.H	collection of features
sipdata_connected_components.C sipdata_connected_components.H	conencted components decomposition of CELs
sipdata_isolated_circles.C sipdata_isolated_circles.H	All circles defined by a single, closed, connected component.

3. Rect_win (rect_win.H) , Sipwin (sipwin.C and sipwin.H) , Sipwin_file_of (sipwin_file_of.C and sipwin_file_of.H)

A `Rect_win` structure defining a rectangular window by specifying its endpoints `x0`, `y0`, `x1` and `y1` and its "hot point" (`xh`, `sub_pixel_x`), (`yh`, `sub_pixel_y`) with is given in sdubpixel coordinates. The class `Sipwin` is derived from `Rect_win` and is used as a container for `Sipdata` objects .

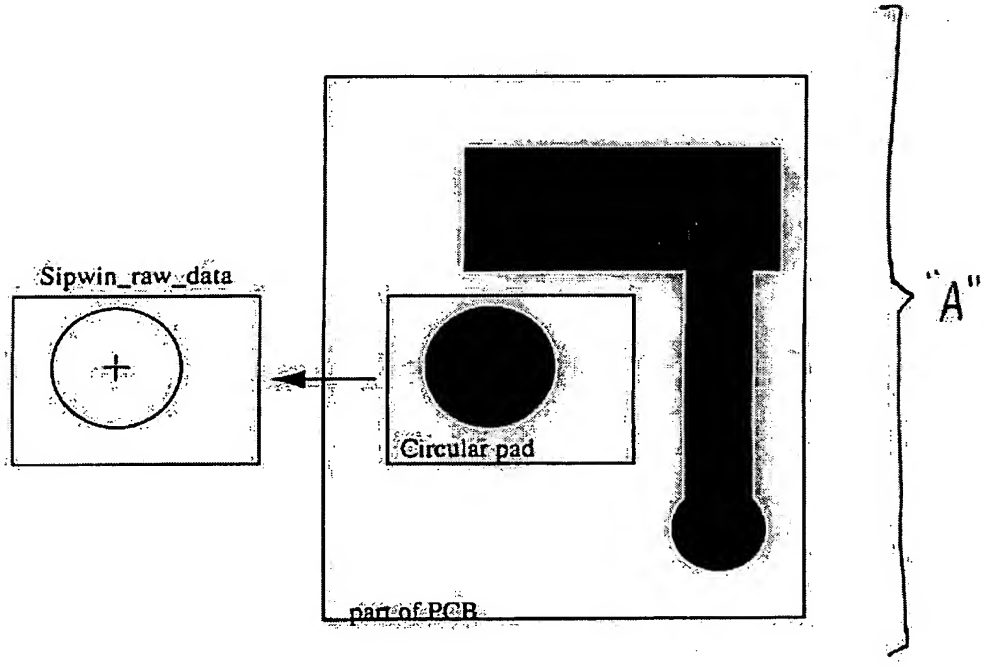
Th public methods of `Sipwin` are:

- `Put( const Sipdata * s)`.
  Put `s` in `Sipwin`. This represents a transfer of ownership from the caller to the window. The `Sipdata` object `s` is given a `const` attribute.
- `PutNonConst( const Sipdata * s )`
  The same as `Put` but `s` is given a non `const` attribute.
- `const Sipdata * Withdraw( int i )`
  `const Sipdata * Withdraw( const char * name )`
  Take an element out from `Sipwin`. This represents a transfer of ownership from `Sipwin` to the caller. The element can be retrieved by its index or by its name. If element not found or if it was found but its attribute is non `const`, then the method returns NULL. In addition we have two methods for taking non `const` elements out of Sipwin. These methods return NULL if element not found or if it was found but its attribute is `const`:
  `Sipdata * WithdrawNonConst( int i )`
  `Sipdata * WithdrawNonConst(const char * name )`
- `const Sipdata * Get( int i )`
  `const Sipdata * Get( const char * name )`
  Get an element from `Sipwin`. The element can be retrieved by its index or by its name. If element not found or if it was found but its attribute is non `const`, then the method returns NULL. In addition we have two methods for getting non `const` elements out of `Sipwin`. These methods return NULL if element not found or if it was found but its attribute is

```
const.
Sipdata * GetNonConst( int i )
Sipdata * GetNonConst(const char * name )
Read.
```
Reads object from file in ASCII or binary form.
```
Write.
```
Writes object to file in ASCII or binary form.
```
Display.
```
Use Display_data object to display graphically the data inside the window.

The `Sipwin_file_of` class is responsible for reading and writing files containing `Sipwindows` from and into files. A file containing collection of `Sipwin` objects can be openned for reading or writing by using the `Open` method. The file can be closed using the `Close` method. The `NumberOfSipDataObjects` method returns the number of `Sipwin` objects in the file. The `Write` method appends a `Sipwin` object to the file and the `Read( int i )` method reads the i'th `Sipwin` object from the file.

The `Sipwin_vector_of` handles a vector containing many `Sipwin_data` elements which are stored in memory. The class provides methods for reading/writing from file (using the `Sipwin_file_of` class) and for fast random access to its entries.



3.1 `Sipwinfunc`, and `Sipwin_vector_of_functions`.

This is the base class for all functions that work on Sipwindows. The class uses a factory (or

virtual constructor) pattern, see also documentation in file sip_factory.H, so that derived classes can be added without need to change existing code. Each object derived from `Sipwinfunc` have a type which is the name of the derived class and a name which uniquely identifies the object among many objects of the same type. The function `SetName` allows to set a name of a `Sipdata` object while the function Name returns that name. The base class `Sipwinfunc` do not have a default constructor. It is necessary to specify a name for the object when constructing it.

Each concrete function contains parameters controling its behaviour. For example, a function that checks circular pad have a parameter specifying the max allowed difference between the radius of the computed pad and the radius of the expected pad. The function will be called with a Sipwin object containing a reference circle and the CELs inside the window. The function will try to fit a circle to given CELs and reports on defect the computed radius is smaller that the expected radius minus tolerance or larger that the expected radius plus tolerance. The function will compute a description of the computed circle and add it to the data of Sipwin.p The base class will contain the following public virtual functions:

- `Type`.
  Returns the name of the derived class.
- `DoCreate`.
  Allocate a derived class object as a pointer to Sipwinfunc.
- `SetParams`.
  Sets parameters for the function from a configuration object. NOte: Each concrete function will have another, overl;oaded, version of SetParams for explicitly setting the parameters for that function.
- `Execute`.
  Perform the computation of the function given a `Sipwin` function as input. Output of computation will be added to this `Sipwin` object.

The class **`Sipwinfunc_vector_of`** is responsible to read a set of functions defined in a configuration file to provide access to the functions etc.

4. Top down methodology, Sipwin top down_reference, `TopDown` data item, top down `Sip_event`, and `TopDown task` and the class `Sipwin_top_down_tests`

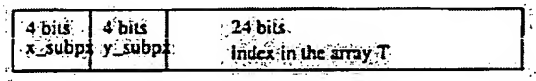
The top down methodology is based on constructing a reference describing the entities to be checked. The reference contains information about the entity (for example, A circular pad of radius of 30 MIL is expected in a given location) and about the test that is to be performed (check that the radius of the pad is not smaller that 25 MIL and not larger than 37 MIL). The creation of a top down reference is very important and may be a difficult task. The reference may be created using accurate CAM data, or it may be learned from a special learn scans. Once a good top down reference is available, it can help considerably to increase the detection and to decrease the amount of false alarms.

The top down reference consists of three files :

- A file containing `Sipwin_data` objects. This file is created by a special learn scan or by processing of CAM data.
- A file containing definiitions of all functions used in the scan with their parameters. For example, if we want to fit circle to CELs with two tolerances then we will have two function definitions in the file.

A file containing a list of function names. The i'th entry in that file contains the name of the function that should be attached to the i'th window from the window file.

A `TopDownReport` data item consists 32 bits devided as follows:



A `Top_down_event` class is derived from `Sip_event` class and contains a single `TopDownReport` object.

The TopDown task loads the top down reference and is responsible for producing the top down data source. If the task works without registration than the TopDown data source consists of top down events which will allow access to the top down table. If there is a registration then the task transform the top down reference from reference coordinates to on-line coordinates and produce the top down data source.

The class `Sipwin_top_down_reference` contains the complete top down reference which consists of a vector of Sipwin_test_raw objects, a vector of Sipwin_test_compound objects, a referenbce table and a top down data source.

The class provides the following public methods

- `LoadConfiguration`.
  -- Use `Sipwin_vector_of` method to load all windows in the file of Sipwindows.
  -- Use `Sipwinfunc_vector_of` to load all functions, with their parameters, from file of functions.
  -- Run over file of function names. Build a vector of pointers to `Sipwinfunc` objects from `Sipwinfunc_vector_of` such that the i'th entry corersponds to the function assotiated with the i'th `Sipwindow`.
  -- Construct a full `Ds_array<Top_down_events>`.
  -- Consider Meta windows. Meta window is a windor that contains data produced by functions operating on other windows. The meta window hirarchy is determined when learning the board. We define to each window, its consumers (if any) and build a data srtucture for defining the consumers of the data produced by the top down function operating on each window. In the meta window itself we shall store information on how many data items it uses.
- `Clear`. Undo LoadConfiguration.
- `Yiterator`.
  Returns iterator to the y'th line of `Ds_array<Top_down_events>`.
- `Sipwinpack * HandleTrigger(const Top_down_event & trigger)`
  The trigger is a top down event , we get the index of the corresponding (the index element of the top down item) Sipwin element in Sipwin_vector_of. The coordinates of ????? its window are transformed from the reference coordinate system to the on-line system.
  Returns NULL on error.

5. Sipwin_trigger_handler

This class is for handling triggers. It consist of the following public methods:

- `LoadConfiguration`.
  Load all information needed for computation.
- `Clear`. Undo LoadConfiguration.
- `Sipwinpack * HandleTrigger(const Sipwin_top_down_reference & td_ref, const Sip_event * trigger)`.
  If the trigger is a top down event ,call `td_ref.HandleTrigger( Top_down_event *)trigger )`. In any other case, use internal logic to create and return required data.

6. Sipwinpack

The class `Sipwinpack` helps that packer program to get information on window in wich data is packed and provide push_back functions to add packed items into Sipdata containers.

The class provides the folowing public methods:

- `Sipwinpack( int x0 int y0, int x1 int y1 )`
- IsOk()
- Win()
- push_back(...)
- NotifyAllFeaturesPacked
- NotifyAllCelsPacked

7. Top_down_task

Is attached to transformation data source and to static top down reference data source. Produces data source of top down events after transformation and ordering of top down events in online liens. Produces alligned top down events.

8. Packer task

Attached to data sources that produce triggers, to top down data source, to cel data source to static top down reference data source and to stating trigger handler data source. Produces queue of windows data source.

The task puts all trigger data into Sip_event lists ordered by priority. Then call trigger handler for each trigger that is not inside existing window. Gets pointers to Sipwinpack objects that are inserted into list of overlapping windows. Packe triggers, data sources and CELs inside Sipwinpack objects containing them. After pack is completed, notify the Sipwinpack object on end of packing and push packed window into queue.

9. test manager task

Uses win_queue data source and static top down reference data source. It pops window from queue, execute the corresponding function and distribute results to costumers. Process defects

if found and push new test into queue if neccessary.

10. Sip_defect and Sip_vector_of_defects

This class will contain description of defects detected during scan. The class is TBD. Probably it will consist of the following

- version number (static attribute of the class)
- on-line --> reference registration (Identity for learn scans)
- Location (x,y) in on-line coordinate system.
- TCL string describing the defect.

The class will provide the following public methods:

- Read/Write to and from file in TCL forms.
- Set/Get for all non static elements.

The class Sip_vector_of_defects will allow read/write of TCL file containing defects and provides iterator to all defects.

Appendix "B"

icpDirectory

```
clyde:~> ls -ltr icp/ovlp/alfi/sip/doc/
total 2076
-rwxr-xr-x    1 shmulik  ws      66560 Oct 29  1996 sip_tests_in_windows.fm*
-rwxr-xr-x    1 shmulik  ws      69632 Aug 21  1997 sip_defect_detection.fm*
-rwxr-xr-x    1 shmulik  ws      92160 Oct 27  1997 sip_future_work.fm*
-rwxr-xr-x    1 shmulik  ws      28672 Oct 27  1997 sip_milestone_aug97.fm*
-rwxr-xr-x    1 shmulik  ws     113664 Oct 27  1997 sip_status.fm*
-rwxr-xr-x    1 shmulik  ws     200704 Jan  5  1998 sip_uguide.fm*
-rwxr-xr-x    1 shmulik  ws     183296 Feb  3  1998 sip_overview.fm*
-rwxr-xr-x    1 shmulik  ws      68608 Feb  3  1998 sip_navigator.fm*
-rwxr-xr-x    1 shmulik  ws      51200 Feb  3  1998 sip_config.fm*
-rwxr-xr-x    1 shmulik  ws       2055 Mar 15  1998 vectorizer_perf*
-rwxr-xr-x    1 shmulik  ws     193262 Mar 15  1998 vec_defl.tif*
-rwxr-xr-x    1 shmulik  ws     203356 Mar 15  1998 vec_maxcomp.tif*
-rwxr-xr-x    1 shmulik  ws     345926 Mar 15  1998 vecperf.tif*
-rwxr-xr-x    1 shmulik  ws        782 Mar 15  1998 vecprf2*
-rwxr-xr-x    1 shmulik  ws         29 Mar 15  1998 vectorizer_rec*
-rwxr-xr-x    1 shmulik  ws         56 Mar 15  1998 vectorizer_rec1*
-rwxr-xr-x    1 shmulik  ws       5389 Mar 30  1998 connectpcb.gif*
-rwxr-xr-x    1 shmulik  ws       6605 Mar 30  1998 connectpcbd0.gif*
-rwxr-xr-x    1 shmulik  ws       6683 Mar 30  1998 connectpcbd1.gif*
-rwxr-xr-x    1 shmulik  ws      66560 Mar 30  1998 connected.fm*
-rwxr-xr-x    1 shmulik  ws      89088 Mar 30  1998 vectorizer.fm*
-rwxr-xr-x    1 shmulik  ws      31744 Nov 15  1998 error_handling_messages.fm*
-rwxr-xr-x    1 shmulik  ws      50176 Jan 19  1999 logger.fm*
-rwxr-xr-x    1 shmulik  ws       3070 Feb 17  1999 install_instructions.txt*
-rwxr-xr-x    1 shmulik  ws      39936 Jul 29  1999 td_reference.fm
-rw-r--r--    1 shmulik  ws      53248 Dec 29  1999 bath_registration.fm
-rw-r--r--    1 shmulik  ws      54272 Dec 29  1999 mechainc_test.fm
-rw-r--r--    1 shmulik  ws       4096 Jan  8  2001 html/
drwxr-xr-x    3 shmulik  ws       4096 Jan  8  2001 cvs/
drwxr-xr-x    2 shmulik  ws
clyde:~>
```

Appendix "C"

24 June 1998
/home/shmulik/icp/alfi/sip/Doc/sip_icp.fm

Orbotech Ltd.

SIP ICP

scenarios

Shmuel Rippa

1. Introduction

In the following we present the SIP scenarios we have in the SIP implementation for the ICP (BGA) machine. We will refer to two subsystems:

- Version I.. The system that we suggest for Sept. 1998.
  Main method of inspection is CEL2VEC for all areas but the Balls area. The inspect will be based on CEL2VEC comparison of reference against online panel without any handling of hardware defects (nick/protrusions), excess/missing or line width defects. The reference created will be as follows:
  -- Front side. Reference consist of Balls, Pads and Voltage lines areas. The reference for Balls consist of description of balls and other references consist of vectorized connected components. Reference will be made for slice 1 (the central camera) only. During the preparation of the reference we shall encode inside the vectorized data also information of different zones (critical Vs noncritical) using the clipping and polyline ID. mechanism.
  -- Rear side. Reference consists of Balls. The reference will consist of the description of the balls. Reference will be made from all slices, transformed by the cameras transformation into one reference system and stored.
  In this version we will not support serious handling of design rule errors. We will not create mask areas and will not filter features inside those areas, by using a full blown filter task. We may wish to define areas to filter and filter out defects falling in those areas by a simlistice algorithm. T
- Version II.
  We will try to make inspection in small areas of interest (ROI) consisting of pads (a window pre pad). The inspection of pads will include (in addition or replacing the CEL2VEC) computation also determining changes in line width of the pad. The Balls area and volatge lines remain as in version I. We shall add to the inspect also windows triggeres by hardware defects, excess/missing and line width defects. The reference for Balls (including the reference for the rear part) and voltage lines will be as in version I. Reference for Pads will be in the form of a window per pad. The reference may incude in addition to vectorized form of the pad also information about width (say, the skeleton of the pad) of the pad. We will have to add a mechanism to corerctly assign tolerances of tests to windows created for defects (nick/protrusions, EM and line width) with respect to their position (critical, non critical etc.).

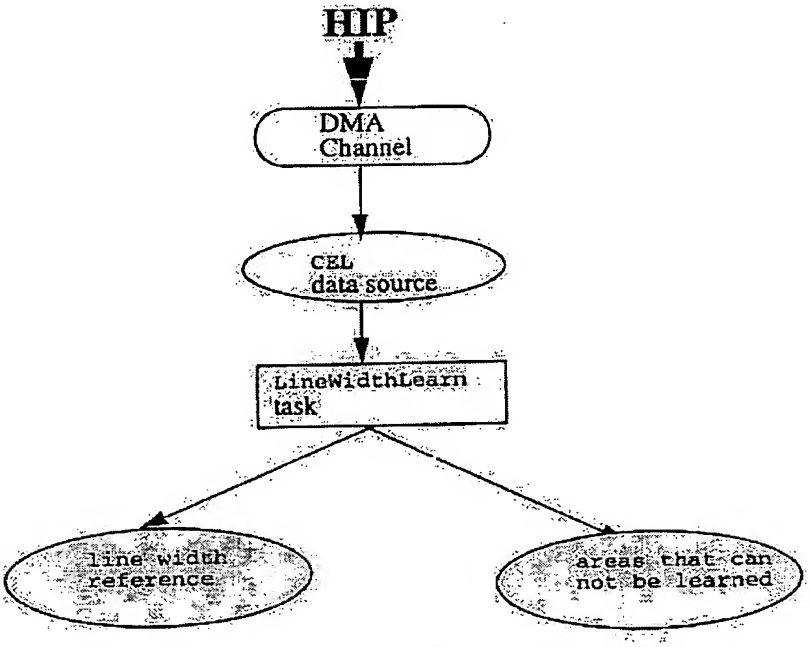
2. Learn line width scenario.

will be used for front side only.

2.1 Version I.

No need for this scenario unless we want to include desing rule errors including min space or min line width.

2.2 Version II



Learn Line Width scenarion

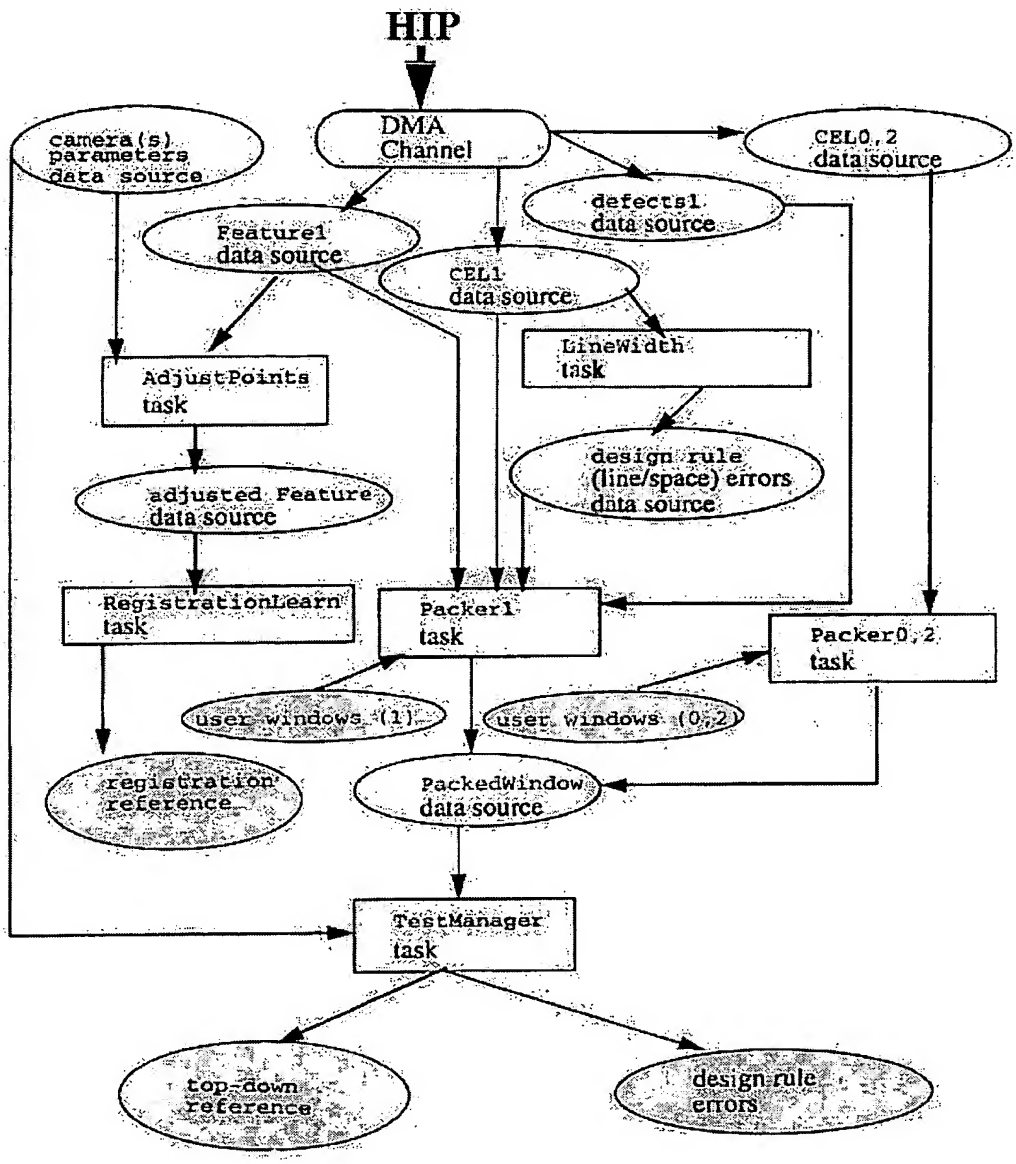
The **LineWidthLearn** task process all CELs and skeleton reports that are used to measure the line width of all "lines" (say, black areas). The line width reference is composed of a set of rectangles. Each rectangle encloses an area on which all line width mesurments had approximately of the same value. In areas of the panel where the widthes of the lines change too much, it is not possible to create a suitable line width reference. The LineWidthLearn task identifies those regions and encloses them in rectangular areas which are produces as areas than can not be learned.

QUESTION - do we want to produce design rule defects? (reduction below min line widh or min space violation?)

3. Learn top down and registration - front side

We shall work on data coming from the central camera only.

3.1 Version I.



CELi : Cels for slice (camera) number i.
Featurei : Features for slice number i.
defectsi : Hardware defects (nick/protrusions) for slice i

camera(s) parameters data source. Provides access to camera(s) parameters. The information includes number of cameras (slices). For each camera we provide the camera lenght

(number of diodes), and an affine transformation that transforms the camera coordinate system into an ideal (mathematical) reference coordinate system.

AdjustPoints task- Takes point data sources (for example, features and defects). Applies the camera transformation and produces the points in the ideal coordinate system. The input to the task may include features (or defects) on more than one slice,. The output is always a single (unsliced) data source. When processing data from more than one slice, the task also eliminates multiply features that are defined on more than one slice on the overlapping area.

RegistrationLearn task- Just takes all features in the ideal coordinate system and store them in the learn reference. If we want to learn only stable features then we must produce these stable features (for example, by the stp top down reference function) and feed them to the registration task (this means that we have to create a new data source of stablke events that are consumed by the RegistrationLearn task). Alternatively, we can get read of this task and produce the stable features directly into the registration reference.

LineWidth task: Measures line and space violations agains fixed line and space criterion. Any violation is reported as line width defect.

AdjustPoints task. Gets the camera parameters. Transform the input (online) features to the ideal coordinate system. Produces transformed features. When the inpout inclides points on more than one slice

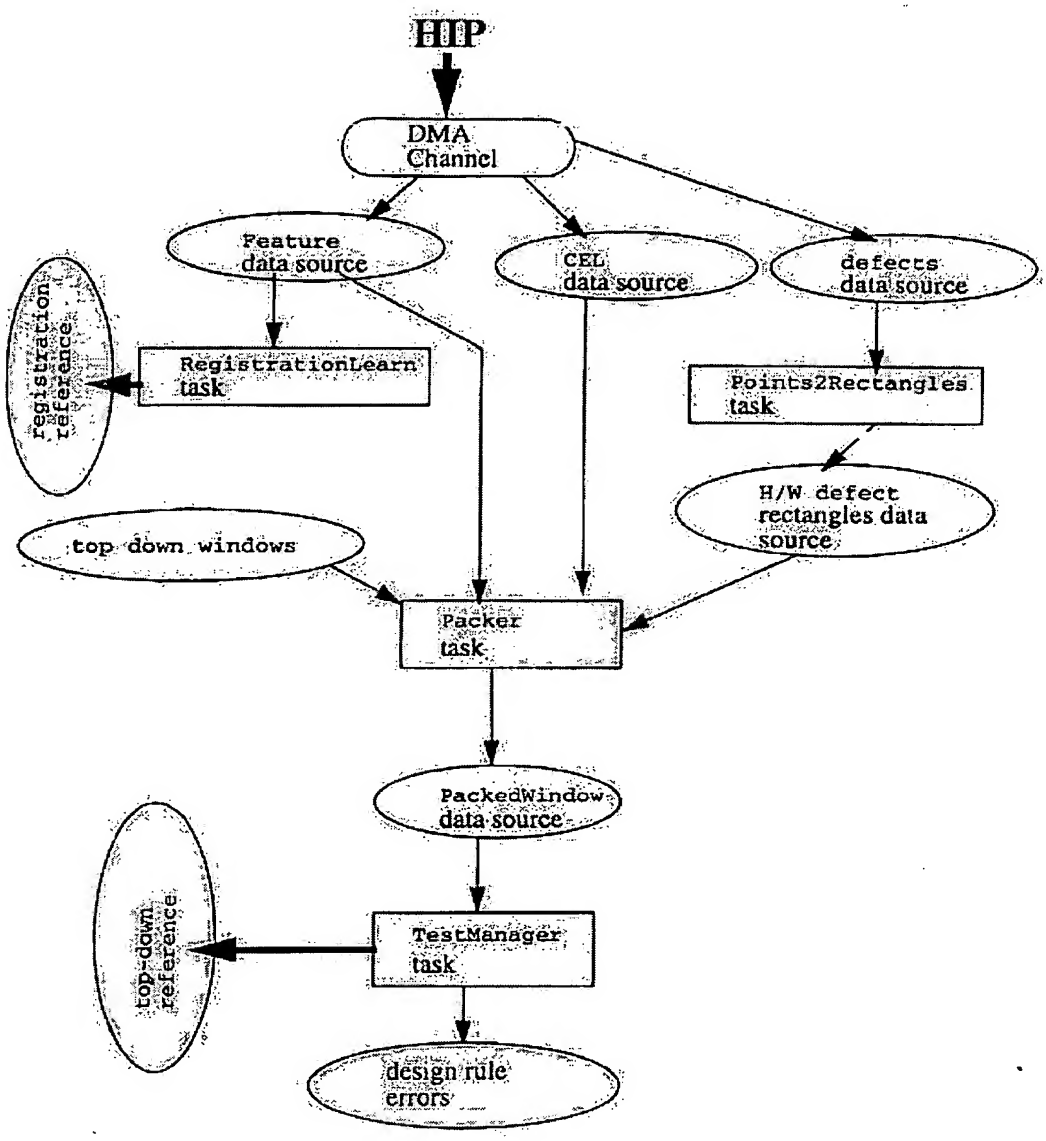
top down windows. consists of
- Balls (user defined). The balls areas.
- Pad areas.
- Voltage lines areas ?
- target areas ?

"E"

PackerTask. Gets one or more rectangles data sources (examples are top down rectangles, width disqualified windows and the rectangles data sources of defects). Applies a simple priority mechanism to dissolve inclusion problems (what i f defect/width disqualified rectangles are included inside top does rectangles, then they are packed inside the top down rectangles). If defect rectangles are found inside disqualified width windows they are packed inside them. All CELs and features are packed inside the requested windows. top down windows and the defect windows that do not fall inside existing top down windows. After windows are packed they are pushed into queue of windows.

TestManager task. Pops widnows from the queue and executed the requested test function that depends on the name of the window:
- Balls. The window is assumed to be composed only of circles where each circle is a single closed connencted component. Design rule checks are performed to ensure that the circles are within permitetd radiuses. Creates a reference of circles.
  Q : What about balls in rear seen by several slices?
  Q : What to do with defect windows found.
- Pad. Determening the critical areas, creating a reference of compressed polygons suitable for CEL2CEL type comparisons. The compressed polygons wil be determined in "large" windows consisting of all user defined windows (pads and voltage lines).
  Q : What to do with defect windows found inside top down windows.

"F"



3.2 Version II.

We shall add to the top down windows also the list of all disqualified windows (that is, windows for which we can not create a line width reference). The test manager task will contain a different function for creating reference for pads. This time we create small reference windows, one around each pad. We need to make sure that the reference windows will allow proper handling of micro registration task. The reference for disqualified width windows will consist of all compressed polygons suitable for CEL2VEC type comparisons.

Q: What to do with defect windows found inside disqualified windows or when disqualified widnows are interscctcd with other top down windows (balls, pads, voltage lines).

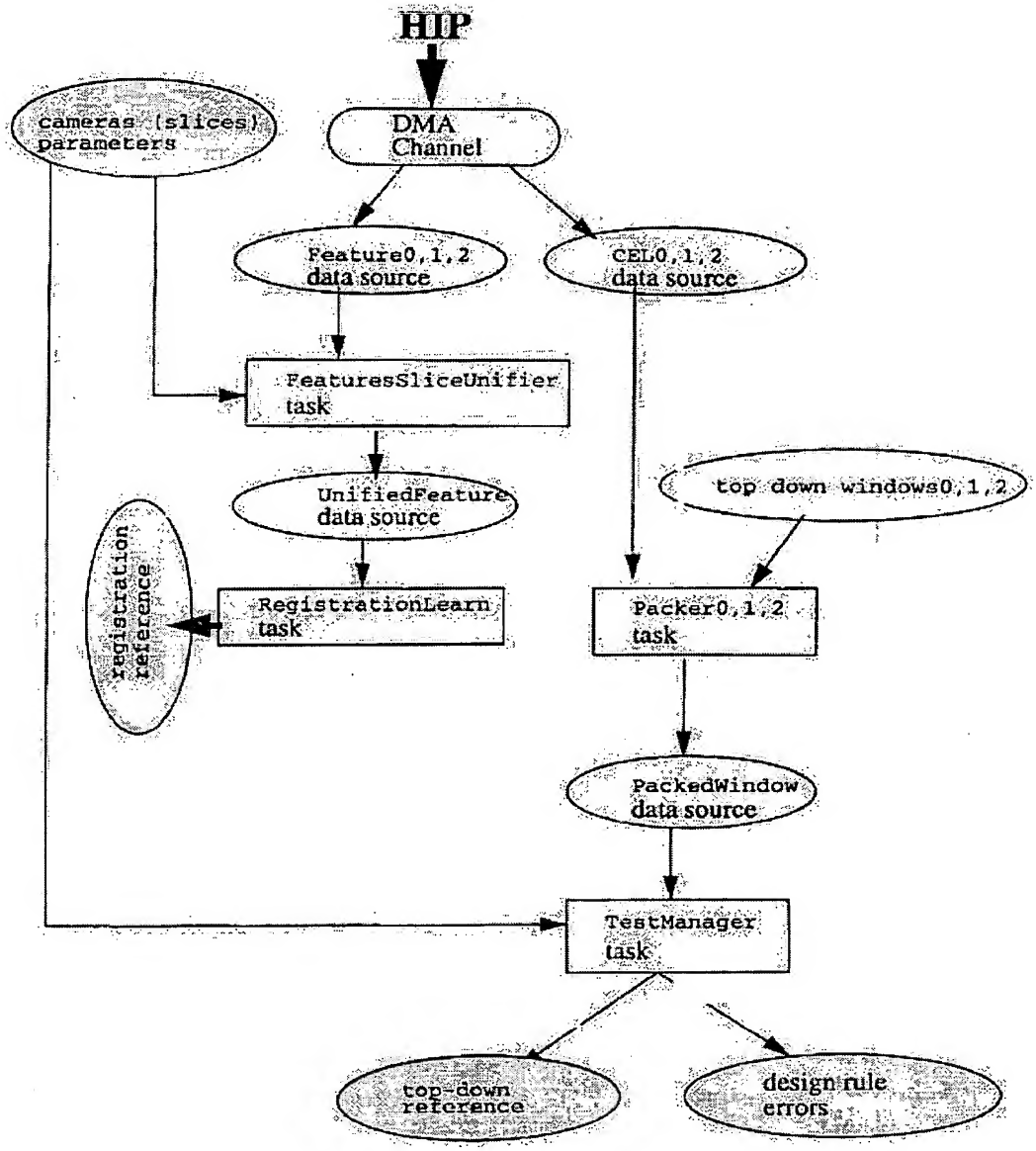
Points2Rectangles. This task consumes data sources consisting of points. It uses some clustering mechanism to group thos points into rectangles stored in *rectangles data source*.

rectangles data source. A data source for rectangles. In order to be able to put a rectangle into this data source, the rectangles should be synamically allocated and point to an object derived from *TypedRectangle* class. A *SmartPtr* to this object will be inserted into the *ds_array* data source. Line Yof this data source will consist of all rectangles of the form [x0,y0] X [x0,Y].

ds_line_collection. This data source is pretty similar to ds_array data source. Th edifefrence is that on any line it store not an array of elements ordered by their x coordinate but a collection of items, not neccessarily ordered. We use push_back operations to insert elements, one by one into the line. We begin the insertion by calling to the comamnd BeginLine() and end it by the method EndLine().

4. Learn top down and registration - back side

4.1 Version I + II



In this scenario we have to handle information coming from three cameras. Each camera is feaded to the hardware which produces slice based data (features, CELs, etc.). We have the *cameras parameters*, that include affine transformations of each camera into one unified coordinate system and information on cameras overlap. We assume that the cameras overlap is larger that the diameter of the largest expected ball. Thre are three data sources for features

(fearure0 for features on first slice, ,feature1 for features on second slice and feature2 for features on third slice , or, in short feature0,1,2) and three data sources fo CELs (CELS0,1,2).

Task *FeaturesSliceUnifier* Takes features from all slices and produce a feature data source on which features are defined by the unified coordinate system. This task takes into account the cameras overlap and merge multiply features detected in two adjacent slices.

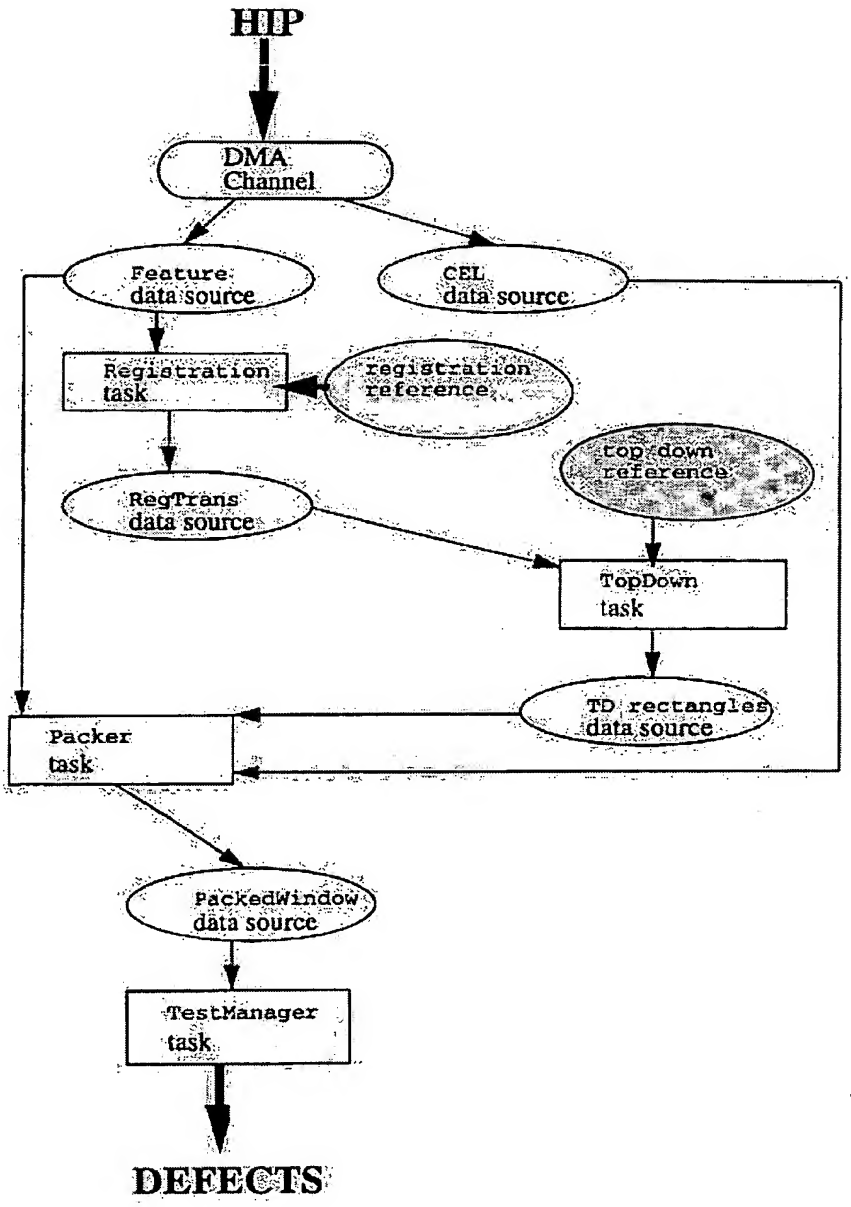
Top down windows include only balls. There are three top down windows of type Balls, one for each slice.

There are three packr tasks, each accepting data from different slice. All packed windows are pushed into one queue The function in test manager task produces one topdown reference on the unified coordinate system after merging multiply balls (falling in two adjacent slices).

5. Inspect execution graph - front side.

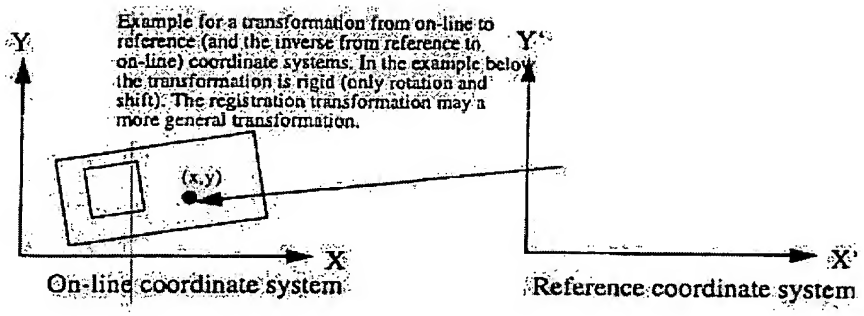
working on central slice only.

5.1 Version I.



Registration task (inspect). Uses features to compute the registration transformation. The transformation maps points from the on-line coordinate system to the reference coordinate system. The registration module also identifies reference features not found in the set of on-line features as missing and on-line features not found in the set of reference features as excess. Excess/missing features are reported as EM defect.

Inspect execution graph - front side.



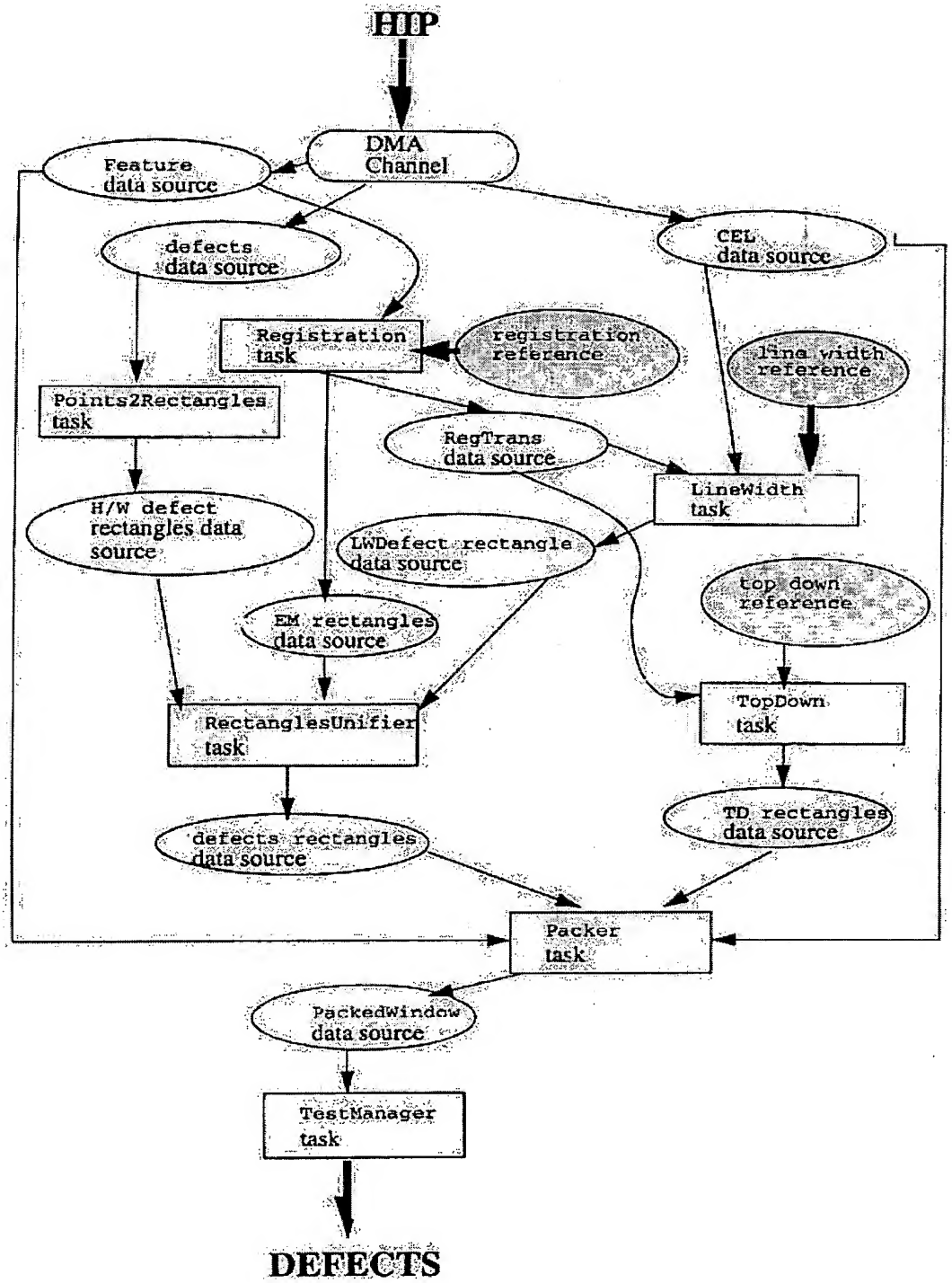
RegTrans data source. A data source that returns for each line y the most appropriate registration transformation for that line.

TopDown task. This task applies the registration transformation to the top down reference (defined in reference coordinate system). It sort transformed trop down windows according to their largest Y coordinate and puts all windows that ends on specified Y coordinate into a rectangles data source.

Test functions include CEL2VEC (micro registration and CEL2VEC comparison) for all Pad and voltage lines and balls comparison for the central balls areas (part design rule and part reference checking).

Inspect execution graph - front side.

5.2 Version II.



The inspect scan scenario uses the following tasks:

LineWidth task (inspect). Computes the width of lines (spaces) in the scanned board. Computed widths are compared to the width of lines (spaces) in the corresponding position on the reference board (the registration transformation is used to find the position of a given computed, width report in the reference coordinate system). Every mismatch between a reference width and an on line width is recorded. The set of LWDefects is clustered and a set of rectangles around the clusters of LWDefects is produced as rectangles data source. Alternativly we may produce the set of LWDefects as points and let the Points2Rectangles task do the clustering.

Registration task. In this version the registration task produces also a list of Excess/Missing features. The list is reported as a set of EM rectangles around EM regions.

RectanglesUnifier task: Accepts various sorts of defect rectangles, cluster them and produce an, hopefully smaller set of defect rectangles.

Packer task. Uses top down rectangles and defect rectangles to define set of windows to test. If a defect window is inside a top down window then the defect rectangle is packed inside the top down window. Otherwise, a defect window is created.

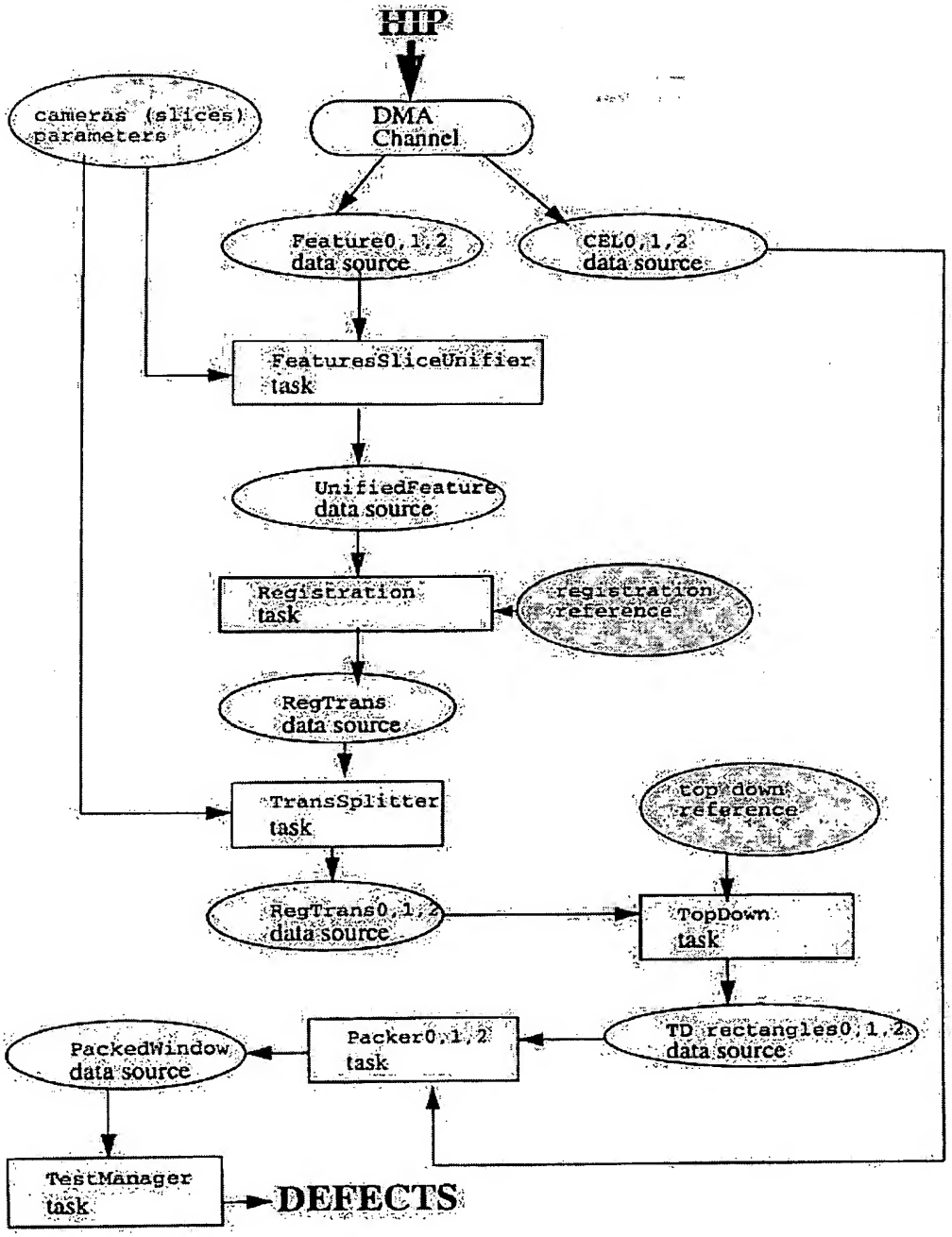
TestManager task. Responsible for the activation of test functions for the various functions. The functions are top down functions that include

- Balls window. Test function as in Version I.
- Pad. Now each pad is serrounded by a small window. That will allow us to combine (or replace) CEL2VEC by a veraity of algorithms, for example line width measurments, etc that are better suited to find the kind of defects that the user request.
- Voltage line. Test as in Version I.
- Each of the above if also includes defect windows.

and defect window that are not included in top down windows. A possible strategy for handling a defect window is the following: If the number of different defects in the window (EM, LW, HW defects) is larger than a prescribed number, then we decide that the window is defective. Otherwise we extract reference data from data base (extraction of CELs from Dbase and vectorization) for CEL2VEC comparison, possibly followed by local line width measurment and other such stuff.

6. Inspect execution graph - back side.

6.1 Version I + II.



Registration task. Gets unified (online) features. Uses the registration reference to computes the registration transformation between the unified online coordinate system and unified reference coordinate system. Produces this transformation in the *RegTrans* data source.

TransSplitter task. Gets online (unified) to reference (unified) transformation and the camera parameters. Compute three different transformations between each slice and the reference (unified) coordinate system. Produces three data sources *RegTrans0,1,2* where RegTransi is the transformation of online slice i into the reference coordinate system.

TopDown task. Uses the top down reference (consisting of a single widow of balls) defined in the (unified) reference coordinate system. Gets the three transformations RegTrans0,1,2. Splits the balls window into three windows, one for each slice. Create three rectangles data sources, one for each slice.

PackerTask. There are three of them, each getting data for differect slice. Packing CELs inside top down rectangles for each slice.

APPENDIX "D"

```
#include "task_packer.H"

#include <cstdio>
#include <cstdlib>
#include <cmath>
using namespace std;

#include "sip_logger.H"
#include "sip_factory.H"
#include "sip_config.H"

#include "sip_general_data.H"
#include "sip_perf_meter.H"
#include "cel_event.H"
#include "cel.H"

#include "sipwinpack.H"

// add a factory for Task_data_transfer to list of factories
//-----------------------------------------------------------------------------
static Register_subclass<Task_packer> r;

// constructor
//-----------------------------------------------------------------------------
Task_packer::Task_packer( )
  : e_windows(NULL), priority_i_line(NULL),
    is_event_marked(NULL)
{
}


// Destructor
//-----------------------------------------------------------------------------
Task_packer::~Task_packer()
{
}

///
const char *   Task_packer::Type() const
{
  return "Task_packer";
}

///
Base_factory *   Task_packer::DoCreate() const
{
  return new Task_packer;
}


// DoLoadConfiguration
//-----------------------------------------------------------------------------
bool Task_packer::DoLoadConfiguration( Sip_config & config )
{
  if ( !packer_helper.Load(config) ) {
    mlog(LOG_DERIVED_ERROR,"DoLoadConfiguration:");
    DoClear();
    return false;
```

```
}

//-------------------------------------------------------------------
// Check that all data sources used and produced are as expected
// For each data source used by packer, set the delay in
// ds_used_space_above vector. This means that the
// task can work on line y only if we have all the lines in data sources
// from  line y up to line y + ds_used_space_above[i_feature]
//-------------------------------------------------------------------
if ( (v_ds_produced.size() == 1) && (v_ds_produced[0]->IsType("Win_queue")) )
{
  ds_queue = (Win_queue *)(v_ds_produced[0]);
}
else
{
  mlog(LOG_APPLIC_ERROR, "DoLoadConfiguration: "
    "Task %s : Illegal list of produced data sources.", Name());
  return false;
}

//-------------------------------------------------------------------
// Setting priorities and size of window
//-------------------------------------------------------------------
max_window_size = packer_helper.max_window_size;
min_priority    = packer_helper.min_priority;
max_priority    = packer_helper.max_priority;
s_width         = max_window_size * (max_priority - min_priority + 1 );
if ( min_priority != 1 )
{
  mlog(LOG_APPLIC_ERROR, "DoLoadConfiguration: "
    "Task %s : min_priority (%d) should be equal to 1."
    ,Name(),min_priority);
  return false;
}
if ( max_priority < min_priority )
{
  mlog(LOG_APPLIC_ERROR,"DoLoadConfiguration: "
    "Task %s: max_priority (%d) should be "
    "greater than or equal to 1."
    ,Name(),max_priority);
  return false;
}
mlog(LOG_STANDARD, "DoLoadConfiguration: "
    "Task %s: min/max_priority        = %d/%d \n "
    "         max_window_size/s_width = %d/%d."
    ,Name(),min_priority,max_priority,max_window_size,s_width);


//-------------------------------------------------------------------
// Search for  Cel data source.
//-------------------------------------------------------------------
int n_used = 0;
int i_cel = IndexOfDataSourceByType(v_ds_used_sync,"Ds_array<Cel>",0);
if ( i_cel == -1 )
{
  ds_cels = NULL;
}
else
```

```
{
  ds_cels = (Ds_array<Cel> *)(v_ds_used_sync[i_cel]);
  ++n_used;
}

//---------------------------------------------------------------------
// Search for  Color_cel data source.
//---------------------------------------------------------------------
int i_color_cel = IndexOfDataSourceByType(v_ds_used_sync,
                                  "Ds_array<Color_cel>",
                                  0);
if ( i_color_cel == -1 )
{
  ds_color_cels = NULL;
}
else
{
  ds_color_cels = (Ds_array<Color_cel> *)(v_ds_used_sync[i_color_cel]);
  ++n_used;
}

//---------------------------------------------------------------------
// Run over all features data sources (if any). For each data source found,
//---------------------------------------------------------------------
ds_features.clear();
ds_features.reserve(10);
int i = -1;
while ((i = IndexOfDataSourceByType(v_ds_used_sync,
                                "Ds_array<Feature>",i+1)) >= 0)
{
  ds_features.push_back( (Ds_array<Feature> *)(v_ds_used_sync[i]) );
  ds_used_space_above[i] = s_width;
  ++n_used;
}

//---------------------------------------------------------------------
// Run over all defects data sources (if any). For each data source found,
//---------------------------------------------------------------------
ds_defects.clear();
ds_defects.reserve(10);
i = -1;
while ((i = IndexOfDataSourceByType(v_ds_used_sync,
                                "Ds_array<Defect>",i+1)) >= 0)
{
  ds_defects.push_back( (Ds_array<Defect> *)(v_ds_used_sync[i]) );
  ds_used_space_above[i] = s_width;
  ++n_used;
}

//---------------------------------------------------------------------
// search for a Width_defect data source. The index of the feature
// in the vector of used data sources is i_width_defect
//---------------------------------------------------------------------
int i_width_defect
  = IndexOfDataSourceByType(v_ds_used_sync,"Ds_array<Width_defect>",0);
if ( i_width_defect == -1 )
{
  ds_width_defect    = NULL;
```

```
}
else
{
  ds_width_defect    =
    (Ds_array<Width_defect> *)(v_ds_used_sync[i_width_defect]);
  ds_used_space_above[i_width_defect] = s_width;
  ++n_used;
}


//-----------------------------------------------------------------
// search for a Color_defect data source. The index of the feature
// in the vector of used data sources is i_color_defect
//-----------------------------------------------------------------
int i_color_defect
  = IndexOfDataSourceByType(v_ds_used_sync,"Ds_array<Color_defect>",0);
if ( i_color_defect == -1 )
{
  ds_color_defect    = NULL;
}
else
{
  ds_color_defect    =
    (Ds_array<Color_defect> *)(v_ds_used_sync[i_color_defect]);
  ds_used_space_above[i_color_defect] = s_width;
  ++n_used;
}


//-----------------------------------------------------------------
// search for Affine2dtrans data source. The index of the feature
// in the vector of used data sources is i_trans
//-----------------------------------------------------------------
int i_trans = IndexOfDataSourceByType(v_ds_used_sync,
                              "Ds_trans<Affine2dtrans>",0);
if ( i_trans == -1 )
{
  ds_trans    = NULL;
}
else
{
  ds_trans    = (Ds_trans<Affine2dtrans> *)(v_ds_used_sync[i_trans]);
  ds_used_space_above[i_trans] = s_width;
  ++n_used;
}

//-----------------------------------------------------------------
// Verify that all used data sources where handled.
//-----------------------------------------------------------------
if ( v_ds_used_sync.size() != (unsigned) n_used )
{
  mlog(LOG_APPLIC_ERROR, "DoLoadConfiguration: "
    "Task %s : "
    "Number of data sources used (%d) is not as expected (%d).",
    Name(),v_ds_used_sync.size(),n_used);
  return false;
}
```

```
  //-----------------------------------------------------------------
  // Make sure that the task is related to a single coordinate system.
  // that is all data sources produced and consumed by the task are
  // from the same coordinate system.
  //-----------------------------------------------------------------
  if ( !(IsUsedCoordinateSystemOk()) )
  {
    mlog(LOG_APPLIC_ERROR, "DoLoadConfiguration: "
       "Task %s : "
       "All used data sources must "
       "be from the same coordinate system. ",
      Name());
    return false;
  }

  if ( !GetCameraAlignedTransformations( UsedCoordinate_System(),
aligned2camera, camera2aligned ) ) {
    mlog(LOG_APPLIC_ERROR,
       "DoLoadConfiguration: "
       "Task %s : "
       "All used data sources must "
       "be from a Camera coordinate system. ",
      Name());
    return false;
  }

  if ( UsedCoordinate_System().Id() < 0 ) {
    mlog(LOG_APPLIC_ERROR,
       "DoLoadConfiguration: "
       "Task %s : "
       "The camera cordinate system of all used data sources must "
       "have a non negative index (current idex = %d). ",
      Name(),UsedCoordinate_System().Id());
    return false;
  }

  // Get client_id from camera_id.
  client_id = (unsigned int)UsedCoordinate_System().Id();

  //-------------------------------------------------------------------------
-----
  // If we arrive safely to this line, it means that the task uses only data
sources
  // from the same camera.
  //-------------------------------------------------------------------------
-----
  //-------------------------------------------------------------------------
-----
  // Allocate memory
  //-----------------------------------------------------------------
  int largest_line_in_scan = Sip_general_data::Largest_line_in_scan(
UsedCoordinate_System() );

  if ( !(largest_line_in_scan > 0) ) {
    mlog(LOG_APPLIC_ERROR, "DoLoadConfig: "
       "Task %s : Cannot handle zero or negative largest line(%d).",
       Name(),largest_line_in_scan);
    return false;
```

```
  }

  max_v_size = largest_line_in_scan + 1;

  events_in_line.reserve(max_v_size);
  for ( int i = 0; i < max_v_size; ++i ) {
    events_in_line.push_back( list<Unified_event>() );
  }

  int max_line_size = Sip_general_data::Largest_pixel_in_line(
UsedCoordinate_System() );

  if ( !(max_line_size > 0) ) {
    mlog(LOG_APPLIC_ERROR, "DoLoadConfig: "
       "Task %s : Cannot handle zero or negative max line size(%d).",
       Name(),max_line_size);
    return false;
  }

  is_event_marked = new bool [max_line_size+1];
  if ( !is_event_marked )
  {
    mlog(LOG_ALLOC_ERROR,"DoLoadConfiguration: "
        "Task %s : Not enough memory.",Name());
    DoClear();
    return false;
  }

  priority_i_line = new int [max_priority + 1];
  if ( !priority_i_line )
  {
    mlog(LOG_ALLOC_ERROR, "DoLoadConfiguration: "
        "Task %s : Not enough memory.",Name());
    DoClear();
    return false;
  }

  int xwidth_of_unrelevant_strip = 20;
  int ywidth_of_unrelevant_strip = 5;
  int cx0 = Sip_general_data::Smallest_pixel_in_line( UsedCoordinate_System() );
  int cx1 = Sip_general_data::Largest_pixel_in_line( UsedCoordinate_System() );
  int cy0 = Sip_general_data::Smallest_line_in_scan( UsedCoordinate_System() );
  int cy1 = Sip_general_data::Largest_line_in_scan( UsedCoordinate_System() );
  relevant_window.x0 = cx0 + xwidth_of_unrelevant_strip;
  relevant_window.x1 = cx1 - xwidth_of_unrelevant_strip;
  relevant_window.y0 = cy0 + ywidth_of_unrelevant_strip;
  relevant_window.y1 = cy1 - ywidth_of_unrelevant_strip;

  return true;
}


// DoClear: Undo DoLoadConfiguration. This function must work even if
// called before DoLoadConfiguration or after error in DoLoadConfiguration
//-----------------------------------------------------------------------
bool Task_packer::DoClear()
{
  packer_helper.Clear();
```

```
  ds_cels             = NULL;
  ds_color_cels       = NULL;
  ds_width_defect     = NULL;
  ds_color_defect     = NULL;
  ds_defects.clear();
  ds_features.clear();
  events_in_line.clear();
  onl_td_events.Clear();

  if ( is_event_marked ) {
    delete [] is_event_marked;
    is_event_marked = NULL;
  }

  if ( priority_i_line ) {
    delete [] priority_i_line;
    priority_i_line = NULL;
  }

  // just to make sure
  if ( e_windows ) {
    delete e_windows;
    e_windows = NULL;
  }

  // clear relevant window
  relevant_window.x0 = 0;
  relevant_window.x1 = 0;
  relevant_window.y0 = 0;
  relevant_window.y1 = 0;

  return true;
}


// DoInitScan
//---------------------------------------------------------------------------
bool Task_packer::DoInitScan()
{
  int largest_line_in_scan = Sip_general_data::Largest_line_in_scan(
UsedCoordinate_System() );
  if ( !onl_td_events.Alloc(largest_line_in_scan+1) )
  {
    mlog(LOG_ALLOC_ERROR, "DoLoadConfiguration: "
        "Task %s : Not enough memory.",Name());
    return false;
  }


  if ( !packer_helper.InitScan() ) {
    mlog(LOG_DERIVED_ERROR,"DoInitScan:");
    return false;
  }

  // assign an intersect line for each priority
  e_windows = new EWindows();
  if ( !e_windows ) {
```

```
    mlog(LOG_ALLOC_ERROR, "DoLoadConfiguration");
    return false;
  }

  for ( int priority = 0; priority <= max_priority; ++priority ) {
    priority_i_line[priority] = e_windows->RequestIntersectLine();
  }
  cels_i_line     = e_windows->RequestIntersectLine();


  first_time_flag = true;
  last_empty_line = -999;

  return true;
}

// undo InitScan
//---------------------------------------------------------------------
bool Task_packer::DoUnInitScan()
{
  if ( e_windows ) {
    delete e_windows;
    e_windows = NULL;
  }
  onl_td_events.Clear();
  packer_helper.UnInitScan();
  return true;
}

//---------------------------------------------------------------------
bool Task_packer::DoProcessLines( int  first_line,
                          int  last_line,
                          bool end_flag)
{
  if ( ds_trans && ds_trans->DsTransIsEmpty() ) {
    mlog(LOG_APPLIC_ERROR, "DoProcessLines: "
       "Task %s : Transformation data source %s is empty "
       " - does not contain any transformation. ",
       Name(),ds_trans->Name());
    return false;
  }

  // get all non fixed top down events (stored in the ref coordinate system)
  // into onl_td_events[y] for all lines y needed by the packer. The range of
  // lines needed is determined by the structure of the packer algorithms (see
  // methods ProcessEventsInFirstLine and ProcessEventsInLine).
  // onl_td_events[y] contains an ordered list of all non fixed top down events
  // (in online coordinate system) defined for line y.
  // the range of lines needed is [td_first_line,td_last_line] and is stored
  // in local variables to allow sanity checks in method
  // FillEventsListFromTopDown.
  int largest_line_in_scan  = Sip_general_data::Largest_line_in_scan(
UsedCoordinate_System() );
  int largest_pixel_in_line = Sip_general_data::Largest_pixel_in_line(
UsedCoordinate_System() );
  packer_helper.ComputeNeededTopDownLines(first_line,last_line,
                                largest_line_in_scan, largest_pixel_in_line,
                                ds_trans, camera2aligned, aligned2camera,
```

```
                                td_first_line, td_last_line,
                                onl_td_events);

// do some special things on the first call on each scan
if ( first_time_flag ) {
  if ( first_line != 0 ) {
    mlog(LOG_APPLIC_ERROR, "DoProcessLines: "
      "Task %s : First line for processing (%d) not equal to 0.",
      Name());
    return false;
  }

  last_empty_line = first_line - 10;

  // Create fixed windows (windows whose locations are fully determined
  // before scan).
  if ( !packer_helper.CreateFixedWindows( last_line, ds_trans, client_id,
                                camera2aligned, aligned2camera,
                                e_windows ) )
  {
    mlog(LOG_DERIVED_ERROR,"DoProcessLines: ");
    return false;
  }

  // process events in first_line (y=0)
  mlog(LOG_ALL, "DoProcessLines: "
     "Processing events in first line (%d).", first_line);
  if ( !ProcessEventsInFirstLine() )
  {
    mlog(LOG_DERIVED_ERROR,"DoProcessLines: ");
    return false;
  }

  // let the loop that follows work on the rest of the lines
  ++first_line;
  first_time_flag = false;
}

// process events for the given range of lines
for ( int y = first_line; y <= last_line; ++y )
{
  // process events line line
  // plog(LOG_ALL,"  Processing events in line %d.\n",y);
  if ( !ProcessEventsInLine(y) )
  {
    mlog(LOG_DERIVED_ERROR,"DoProcessLines: ");
    return false;
  }
}

if ( end_flag )
{
  // make sure to release all windows remaining in e_windows
  if ( !ProcessLastLine(last_line) )
  {
    mlog(LOG_DERIVED_ERROR,"DoProcessLines: ");
    return false;
  }
```

```
  }

  return true;
}


//-----------------------------------------------------------------------
// Handle all side effects of task during scan:
// free allocations made during scan close files created during scan, etc.
//-----------------------------------------------------------------------
bool Task_packer::DoEndScan()
{
  if (e_windows) {
    mlog(LOG_ALL, "DoEndScan: "
       "End of Scan for task %s. Number of unreleased (un tested) "
       "windows is %d.",
       Name(),e_windows->size());
  } else {
    mlog(LOG_APPLIC_ERROR,"DoEndScan: e_windows is NULL.");
    return false;
  }
  return true;
}



//-----------------------------------------------------------------------
//  process events for first line (y=0). Needs all events on lines
//  y = 0,...,s_width = max_window_size * (max_priority - min_priority + 1 ).
//-----------------------------------------------------------------------
bool Task_packer::ProcessEventsInFirstLine()
{
  // Create ALL event windows (of all priorities) for lines
  // y = 0 ,.., max_window_size. This is accomplished by creating event
  // windows for
  // (1) all events of the max priority for lines
  //     y = 0 ,..., s_width
  // (2) all events of priority (max_priority - 1) for lines
  //     y = 0 ,..., s_width - max_window_size
  // .
  // .
  // .
  // () all events of the min priority (== 1) for lines
  //    y = 0 ,..., max_window_size.
  //
  // NOTE : During the computation, lists of events of the given
  // priority are created. The lists of events of a given priority
  // on a given line y is merged into the list of all events for
  // that line (stored in the list events_in_line[y]
  //-----------------------------------------------------------------------
  for ( int priority = max_priority; priority >= min_priority; --priority )
  {
    for ( int y = 0; y <= max_window_size*(priority-min_priority+1); ++y )
    {
      // NOTE (side effects)
      // 1. Intersect_line for priority is moved to line y
      // 2. event_line for line y is updated to include all
      //    events of priority 'priority'.
```

```
      // 3. create new event windows for events of priority
      //    'priority' that are not inside previously created events.
      //-------------------------------------------------------------
      if ( !CreateEventWindowsForPriority(y, priority) )
      {
      mlog(LOG_DERIVED_ERROR,"ProcessEventsInFirstLine: ");
      return false;
      }
    }
  }

  // add events of no priority to list of events for lines
  // y = 0,...,max_window_size
  //-------------------------------------------------------------
  for ( int y = 0; y <= max_window_size; ++y )
  {
    if ( !AddEventsOfNoPriorityToEventsLine(y) )
    {
      mlog(LOG_DERIVED_ERROR,"ProcessEventsInFirstLine.");
      return false;
    }
  }

  // pack all events intersecting line y into windows containing them
  if ( !PackInLine(0) ) {
    mlog(LOG_DERIVED_ERROR,"ProcessEventsInFirstLine: ");
    return false;
  }

  // List of events for first line is no longer needed - clear it !!
  events_in_line[0].clear();

  return true;
}


//-----------------------------------------------------------------------------
//  Process all events in line y.
//  NOTE : The first call to this function must be with y = 1.
//         The n'th call to this routine mustr have y = n .
//-----------------------------------------------------------------------------
bool Task_packer::ProcessEventsInLine( int y )
{
  //
  //  Create ALL event windows (of all priorities) for line y.
  //  This is accomplished by creating the event windows for
  //  (1) all events of the max priority for line y + s_width
  //  (2) all events of priority (max_priority - 1) for line
  //      y + s_width - max_window_size.
  //  .
  //  .
  //  .
  //  () all events of the min_priority (== 1) for line
  //     y + max_window_size.
  //
  //  NOTE : During the computation, lists of events of the given
  //  priority are created. The lists of events of a given priority
  //  on a given line y is merged into the list of all events for
```

```
  //  that line (stored in the list pointed to by & ((ev_data->events)[y]))
  //-------------------------------------------------------------------------
  for ( int priority = max_priority; priority >= min_priority; priority-- )
  {
    //
    //  NOTE (side effects)
    //  1. Intersect_line for priority is moved to line line
    //  2. event_line for line 'line' is updated to include all
    //     events of priority 'priority'.
    //  3. create new event windows for events of priority
    //     'priority' that are not inside previously created events.
    //-------------------------------------------------------------
    int              line = y + max_window_size*(priority-min_priority+1);
    if ( !CreateEventWindowsForPriority(line, priority) )
    {
      mlog(LOG_DERIVED_ERROR,"ProcessEventsInLine: ");
      return false;
    }
  }

  // add events of no priority to list of events for line
  // y + max_window_size
  //-------------------------------------------------------------------
  if ( !AddEventsOfNoPriorityToEventsLine(y + max_window_size) )
  {
    mlog(LOG_DERIVED_ERROR,"ProcessEventsInLine: ");
    return false;
  }

  // pack all events and C#ls in line y into windows containing them
  if ( !PackInLine( y ) )
  {
    mlog(LOG_DERIVED_ERROR,"ProcessEventsInLine: ");
    return false;
  }

  // clear list of events for line y which is no longer needed
  events_in_line[y].clear();

  return true;
}


//-------------------------------------------------------------------------
//  create event windows from events of the given priority on line y.
//  Does the following:
//    (1) Produces a list of events of the given priority. Mark all events
//        in this list.
//    (2) Moves intersect line assotiated with given priority to line y.
//    (3) Unmark all events from the list of events which are 'inside'
//        one or more (previously created) event windows.
//    (4) Create new event windows from all events that remain marked.
//    (5) Add list of events of the given priority to list of all events
//        in line y.
//
//    NOTE : validity of input parameters is not tested.
//-------------------------------------------------------------------------
bool Task_packer::CreateEventWindowsForPriority(int y,int priority)
```

```
{
  if ( y > Sip_general_data::Largest_line_in_scan( UsedCoordinate_System()) ) {
    return true;
  }

  // This list will contain all event of the given priority on line y
  //-------------------------------------------------------------------
  list<Unified_event> list_of_events;

  // produce list_of_events: all events of priority `priority' in line y
  // all elements of is_event_marked corersponding to list_of_events are set
  // to true (these events are marked)
  //-------------------------------------------------------------------
  if ( !CreateListOfEvents( list_of_events, is_event_marked, y, priority ) )
  {
    // could not produce list of events of the given priority
    mlog(LOG_DERIVED_ERROR,"CreateEventWindowsForPriority: ");
    return false;
  }

  // do the following only if the list of events created is not empty
  //-------------------------------------------------------------------
  if ( !list_of_events.empty() )
  {
    // Moves intersection line connected to events of given priority
    // to line y
    //-----------------------------------------------------------------
    int index_i_line = priority_i_line[priority];
    if ( !e_windows->MoveIntersectLineToLine(index_i_line, y) )
    {
      mlog(LOG_APPLIC_ERROR, "CreateEventWindowsForPriority: "
        "Task %s : Failed to move intersect line to line %d.",
        Name(),y);
      return false;
    }

    // unmark all events of line y that are 'inside' current event windows
    UnmarkEventsInsideWindows(list_of_events,is_event_marked, y, index_i_line);

    // create event windows from all marked events
    if ( !AddWindowsForMarkedEvents( list_of_events, is_event_marked, y ) )
    {
      mlog(LOG_DERIVED_ERROR,"CreateEventWindowsForPriority: ");
      list_of_events.clear();
      return false;
    }

    // merge list_of_events of priority 'priority' into line of events
    // containing all events on line y.
    //-----------------------------------------------------------------
    events_in_line[y].merge( list_of_events );
  }

  return true;
}

//-------------------------------------------------------------------------
```

```
//  Create events of no (zero) priority for line y and merge them with
//  all events already defined for line y.
//-------------------------------------------------------------------------
bool Task_packer::AddEventsOfNoPriorityToEventsLine(int y )
{
  if ( y > Sip_general_data::Largest_line_in_scan( UsedCoordinate_System()) ) {
    return true;
  }

  // This list will contain all event of the given priority on line y
  list<Unified_event> list_of_events;
  int                 no_priority = 0;

  // produce list_of_events containing all events of no priority on line y
  if ( !CreateListOfEvents( list_of_events, is_event_marked, y, no_priority ) )
  {
    // could not produce list of events of the given priority
    mlog(LOG_DERIVED_ERROR,"AddEventsOfNoPriorityToEventsLine: ");
    return false;
  }

  // merge events of no priority with the rest of the events in line y
  events_in_line[y].merge( list_of_events );

  return true;
}



// Pack events and CELs in line y
//-------------------------------------------------------------------------
bool Task_packer::PackInLine( int y)
{
  // Moves cels/features intersection line connected to line y
  //-----------------------------------------------------------------
  if ( !e_windows->MoveIntersectLineToLine(cels_i_line, y) )
  {
    mlog(LOG_APPLIC_ERROR,"PackInLine: "
       "Failed to move intersect line to line %d.",y);
    return false;
  }

  PackEvents(cels_i_line, events_in_line[y]);
  PackCels(y,cels_i_line,ds_cels);
  PackColorCels(y,cels_i_line,ds_color_cels);

  // Run over all windows having Y1 coordinate equal to y (this list
  // is produced as a side effect of MoveIntersectLineToLine).
  // Notify on end of packing and push each packed window into the
  // queue.
  //-------------------------------------------------------------------------
  if ( !HandleReleasedWindows(y) ) {
    mlog(LOG_DERIVED_ERROR,"PackInLine: ");
    return false;
  }

  return true;
}
```

```
// Release all windows remaining in system at end of scan.
//-------------------------------------------------------------------------
bool Task_packer::ProcessLastLine(int last_line)
{
  // Moves cels/features intersection line connected to line y
  //-----------------------------------------------------------------
  if ( !e_windows->MoveIntersectLinePastLastLine(cels_i_line) )
  {

    mlog(LOG_APPLIC_ERROR, "ProcessLastLine: "
        "Failed to move intersect line past last.");
    return false;
  }

  // Run over all remaining windows in the system
  // Notify on end of packing and push each packed window into the
  // queue.
  //-------------------------------------------------------------------------
  if ( !HandleReleasedWindows(last_line) )
  {
    mlog(LOG_DERIVED_ERROR,"ProcessLastLine: ");
    return false;
  }
  return true;
}


// Handle windows released after intersect cels_i_line line is moved
// to some line.
//-------------------------------------------------------------------------
bool Task_packer::HandleReleasedWindows(int y_release )
{
  // Run over the list of all windows having the same Y1 coordinate.
  // this list
  // is produced as a side effect of MoveIntersectLineToLine).
  // Strangely, the same Y coordinate is equal to y_release-1 unless
  // we got to the last window.
  // Notify on end of packing and push each packed window into the
  // queue.
  //-------------------------------------------------------------------------
  vector<Sipwin *> v_out_windows;
  int              y_of_line       = y_release;
  bool             is_empty_line   = true;

  e_windows->Lbegin();
  while ( !(e_windows->Lend())  ) {
    Event_window * ew = e_windows->Literator();
    Sipwinpack * winpack = (Sipwinpack *)(ew->win);
    // end of packing for winpack
    unsigned int n_events            = 0;
    unsigned int n_cels              = 0;
    unsigned int n_skel              = 0;
    unsigned int n_color_cels        = 0;
    unsigned int n_color_junctions   = 0;

    if ( !(winpack->NotifyAllUnifiedEventsPacked(n_events)) ) {
```

```
      mlog(LOG_DERIVED_ERROR,"HandleReleasedWindows:");
      return false;
    }

    if ( ds_cels ) {
      if ( !(winpack->NotifyAllCelsPacked(n_cels,n_skel)) ) {
      mlog(LOG_DERIVED_ERROR,"HandleReleasedWindows:");
      return false;
      }
    }

    if ( ds_color_cels ) {
      if ( !(winpack->NotifyAllColorCelsPacked(n_color_cels,n_color_junctions))
 ) {
      mlog(LOG_DERIVED_ERROR,"HandleReleasedWindows:");
      return false;
      }
    }

    // transfer window (including transfer of ownership)
    // from winpack to to queue. Then delete winpack that is no longer
    // needed.
    if ( winpack->do_pack_and_process ) {
      Sipwin * w = winpack->WithdrawWin();
      if ( w ) {
      mlog(LOG_ALL,
           "HandleReleasedWindows: window %s (%d,%d)-->(%d,%d) is packed "
           "with \n  %d CELs \n  %d colour_cels \n  %d unified events.",
           w->Type(),w->X0(),w->Y0(),w->X1(),w-
>Y1(),n_cels,n_color_cels,n_events);

      v_out_windows.push_back( w );
      if ( is_empty_line ) {
        y_of_line       = w->Y1();
        is_empty_line   = false;
      }
      else {
        if ( w->Y1() != y_of_line ) {
          mlog(LOG_WARN,
             "HandleReleasedWindows: Y!(%d) of window is different from y1(%d)
of another "
             "window. ",w->Y1(),y_of_line);
        }
      }
      //    fprintf(stderr," y_release = %d, Y1 = %d winpack(y1)=
%d\n",y_release,w->Y1(),winpack->y1);
      //      ds_queue->Push( winpack->WithdrawWin() );
      }
    }
    delete winpack;


    // It is imporntant to call e_windows->Lnext() BEFORE a call to
    // e_windows->RemoveEventWindow(ew). To understand why we need to
    // know how event windows are placed in linked lists. Each event
    // window have a 'next' variable which points to the next event
    // window in some list. Such a list is th elinked list created by
    // the method MoveIntersectLineToLine() which contains all windows
```

```
    // having the given y coordinate as their last (y1) coordinate.
    // Another list is a list of 'free' event windows. The event window
    // is added to this list after a call to e_windows->RemoveEventWindow.
    // and after that call the 'next' variable of the removed event window
    // will no longer point to the next element in the linked list.
    // Thus we move to the next element of the linked list BEFORE we
    // call to e_windows->RemoveEventWindow() by calling to e_windows->Lnext().
    //-------------------------------------------------------------------------
    e_windows->Lnext();

    if ( !e_windows->RemoveEventWindow(ew) )
    {
      mlog(LOG_ALLOC_ERROR, "HandleReleasedWindows: "
        "Failed to event window [%d,%d]x[%d,%d].",
            ew->x0,ew->x1,ew->y0,ew->y1);
      return false;
    }
  }

  if ( is_empty_line ) {
    if ( (last_empty_line+1) == y_release ) {
      // previous line was also an empty line, put previous as empty line
      if ( !ds_queue->Push_y( v_out_windows, last_empty_line ) ) {
      mlog(LOG_DERIVED_ERROR,
           "HandleReleasedWindows: Task %s, last_empty_line=%d ",
           Name(),last_empty_line);
      return false;
      }
    }
    last_empty_line = y_release;
  }
  else {
    // non empty line
    if ( !ds_queue->Push_y( v_out_windows, y_of_line ) ) {
      mlog(LOG_DERIVED_ERROR,
         "HandleReleasedWindows: Task %s, y_release=%d ",
         Name(),y_release);
      return false;
    }
  }

  return true;
}


// Create a list of events of the given priority. Mark all created
// events. (set value of the coresponding is_event_marked to true)
//-------------------------------------------------------------------------
bool
Task_packer::CreateListOfEvents( list<Unified_event> & events,
                         bool * is_event_marked,
                         int y,
                         int priority)
{
  // This list will contain events from given data source on line y
  list<Unified_event> list_of_events;

  // add events of given priority from feature data source
```

```
  for ( unsigned int i = 0; i < ds_features.size(); ++i )
  {
    if (!FillEventsListFromFeatures(list_of_events,ds_features[i],y,priority))
    {
      mlog(LOG_DERIVED_ERROR,"CreateListOfEvents: ");
      return false;
    }
    events.merge( list_of_events );
  }

  // add events of given priority from feature data source
  for ( unsigned int i = 0; i < ds_defects.size(); ++i )
  {
    if ( !FillEventsListFromDefects(list_of_events,ds_defects[i],y,priority) )
    {
      mlog(LOG_DERIVED_ERROR,"CreateListOfEvents: ");
      return false;
    }
    events.merge( list_of_events );
  }

  // add events of given priority from width defects data source
  if (
!FillEventsListFromWidthDefects(list_of_events,ds_width_defect,y,priority) )
  {
    mlog(LOG_DERIVED_ERROR,"CreateListOfEvents: ");
    return false;
  }

  // add events of given priority from width defects data source
  if (
!FillEventsListFromColorDefects(list_of_events,ds_color_defect,y,priority) )
  {
    mlog(LOG_DERIVED_ERROR,"CreateListOfEvents: ");
    return false;
  }

  events.merge( list_of_events );

  // Add top down events of the given priority
  if ( !FillEventsListFromTopDown(list_of_events, y, priority) )
  {
    mlog(LOG_DERIVED_ERROR,"CreateListOfEvents: ");
    return false;
  }
  events.merge( list_of_events );

  if ( events.size() > 0 )
  {
    plog(LOG_ALL,
             "Line %8d (priority %3d ) #events %5d.\n",
             y,priority,events.size());
  }

  // mark all events (of given priority) in line.
  for ( unsigned int i = 0; i < events.size(); ++i )
  {
    is_event_marked[i] = true;
```

```
  }

  return true;
}

// Create a list of events of the given priority from feature data
// sopurce.
//-------------------------------------------------------------------------
bool
Task_packer::FillEventsListFromFeatures(list<Unified_event> & list_of_events,
                                Ds_array<Feature>    * ds,
                                int y,
                                int priority)
{
  if ( !ds ) return true;

  list_of_events.clear();

  // run over all features with the given priority
  if ( priority == packer_helper.feature_priority ) {
    // iterators over features (triggers)
    Ds_array_iter<Feature>           iter( ds );
    Feature                        * element;
    Unified_event ev;

    ev.type = Unified_event::FEATURE;
    ev.y    = y;

    // loop over all features of line y
    iter.BeginLine( y );
    while ( (element = iter.Next()) )
    {
      ev.x              = element->x;
      ev.feature        = *element;
      // take feature to list only if it is inside the relevant area
      if ( (ev.x > relevant_window.x0) && (ev.x < relevant_window.x1) ) {
      list_of_events.push_back( ev );
      }
    }
  }
  return true;
}

// Create a list of events of the given priority from defect data
// sopurce.
//-------------------------------------------------------------------------
bool
Task_packer::FillEventsListFromDefects(list<Unified_event> & list_of_events,
                                Ds_array<Defect >      * ds,
                                int y,
                                int priority)
{
  if ( !ds ) return true;

  list_of_events.clear();

  // run over all defects with the given priority
```

```
  if ( priority == packer_helper.defect_priority ) {
    // iterators over defects (triggers)
    Ds_array_iter<Defect>           iter( ds );
    Defect                        * element;
    Unified_event ev;

    ev.type = Unified_event::DEFECT;
    ev.y    = y;

    // loop over all defects of line y
    iter.BeginLine( y );
    while ( (element = iter.Next()) )
    {
      ev.x            = element->x;
      ev.defect       = *element;
      // take defect to list only if it is inside the relevant area
      if ( (ev.x > relevant_window.x0) && (ev.x < relevant_window.x1) ) {
      list_of_events.push_back( ev );
      }
    }
  }
  return true;
}


// Create a list of events of the given priority from width defect data
// sopurce.
//-----------------------------------------------------------------------
bool
Task_packer::FillEventsListFromWidthDefects(list<Unified_event> &
list_of_events,
                                    Ds_array<Width_defect> * ds,
                                    int y,
                                    int priority)
{
  if ( !ds ) return true;

  list_of_events.clear();

  // run over all features with the given priority
  if (priority == packer_helper.width_defect_priority ) {
    // iterators over features (triggers)
    Ds_array_iter<Width_defect>      iter( ds );
    Width_defect                   * element;
    Unified_event ev;

    ev.type = Unified_event::WIDTH_DEFECT;
    ev.y    = y;

    // loop over all features of line y
    iter.BeginLine( y );
    while ( (element = iter.Next()) )
    {
      ev.x             = element->x;
      ev.width_defect  = *element;
      // take width defect to list only if it is inside the relevant area
      if ( (ev.x > relevant_window.x0) && (ev.x < relevant_window.x1) ) {
      list_of_events.push_back( ev );
```

```
      }
    }
  }
  return true;
}


// Create a list of events of the given priority from color defect data
// sopurce.
//-----------------------------------------------------------------------
bool
Task_packer::FillEventsListFromColorDefects(list<Unified_event> &
list_of_events,
                                  Ds_array<Color_defect>      * ds,
                                  int y,
                                  int priority)
{
  if ( !ds ) return true;

  list_of_events.clear();

  // run over all defects with the given priority
  if ( priority == packer_helper.defect_priority ) {
    // iterators over defects (triggers)
    Ds_array_iter<Color_defect>     iter( ds );
    Color_defect                  * element;
    Unified_event ev;

    ev.type = Unified_event::DEFECT;
    ev.y    = y;

    // loop over all defects of line y
    iter.BeginLine( y );
    while ( (element = iter.Next()) )
    {
      ev.x            = element->x;
      ev.defect.x     = element->x;
      ev.defect.data  = Defect::ConvertToDefect(*element);
      // take color defect to list only if it is inside the relevant area
      if ( (ev.x > relevant_window.x0) && (ev.x < relevant_window.x1) ) {
      list_of_events.push_back( ev );
      }
    }
  }
  return true;
}


// Create a list of events of the given priority from top down events on line y
// This list is the list onl_td_events[y]
//-----------------------------------------------------------------------
bool
Task_packer::FillEventsListFromTopDown(list<Unified_event> & list_of_events,
                                int y,
                                int priority)
{
  list_of_events.clear();
```

```
  if ( priority == packer_helper.td_priority ) {
    if ( (y < td_first_line) || (y > td_last_line) )
    {
      mlog(LOG_APPLIC_ERROR, "FillEventsListFromTopDown: "
        "Task %s : Trying to use top down events from line %d. "
        "(top down events are defined only for lines "
        "[%d,%d]", Name(), y, td_first_line, td_last_line);
      return false;
    }

    // move y'th line (containing sorted list of TD events for that line)
    // into list of events.
    //-------------------------------------------------------------------
    list_of_events.splice(list_of_events.end(),onl_td_events[y]);
  }

  return true;
}


// Unmark events that are inside existing windows that intersect the
// intersection line number index_i_line.

//-------------------------------------------------------------------------
void
Task_packer::UnmarkEventsInsideWindows( list<Unified_event> & events,
                                        bool * is_event_marked,
                                        int y,
                                        int index_i_line)
{
  Intersect_line  * i_line    = e_windows->GetIntersectLine(index_i_line);

  // return if list_events is empty
  //-----------------------------------------------------------------------
  if ( events.empty() )    return;

  // iterators over list of events. it will point to event on coordinate
  // ((*it).x,y)
  //-----------------------------------------------------------------------
  unsigned int i_event     = 0;
  list<Unified_event>::iterator it       = events.begin();
  list<Unified_event>::iterator list_end = events.end();

  // mark all events to be candidates for trigegrs for new windows
  for ( unsigned int i = 0; i < events.size(); ++i ) {
    is_event_marked[i_event] = true;
  }


  // If no event windows covering line y, Check if some events in list
  // are covered by previous events in list
  if ( i_line->empty() ) {
    UnmarkEventsInLine(events,is_event_marked);
    return;
  }


  // Move to first Xendpoint. Xendpoints iterator will point at an Xendpoint
```

```
// with coordinate i_line->X(). The list of overlapping windows will
// contain all event windows overlapping the segment between i_line->X()
// and the Xendpoint with smaller X coordinate
//-------------------------------------------------------------------------
i_line->Xbegin();

// while not_end_of_Xendpoints of intersected event windows
// and not_end_of_list_of_events.
//-------------------------------------------------------------------------
while ( !(i_line->Xend()) ) {
  // Check if there are overlapping event window.
  unsigned int osize = i_line->osize();
  while ( (*it).x < i_line->X() ) {
    if ( !(packer_helper.IgnoreInclusionZone(*it)) ) {
    //  event *it is not in ignore mode, that is if it is included
    // in a previous defined window, then no it will not be a trigger
    // for window openning.
    // Check if current_event is inside a set of overlapping event_windows.
    int x = (*it).x;
    // check if event is inside existing overlapping windows (openned
    // on previous lines
    for (unsigned int iosize = 0; iosize < osize; ++iosize) {
      Sipwinpack   *  w
        = (Sipwinpack *)(i_line->GetOvelappingWindow(iosize));

      //  If event is "contained" in some event window, unmark it so that
      //  it will not trigger the openning of new event window.
      if ( (y >  w->izone_y0) && (y < w->izone_y1)
           &&
           (x >  w->izone_x0) && (x < w->izone_x1)  ) {
        // event *it is inside the i'th window
        is_event_marked[i_event] = false;
        break;
      }
    }
    }

    ++it;
    ++i_event;
    if ( !(it != list_end) ) {
    // end of list of events. Check if some marked
    // events can be unmarked because they are inside windows
    // created by previous unmarked events from the same line.
    UnmarkEventsInLine(events,is_event_marked);
    return;
    }
  }

  // (*it).x >= i_line->X() (Possibly we passed past the last event
  // in list. Move to next Xendpoint.
  //-----------------------------------------------------------------------
  i_line->Xnext();
}

// Try to unmark more marked events. Such
// events can be unmarked because they are inside windows
// created by previous unmarked events from the same line.
UnmarkEventsInLine(events,is_event_marked);
```

```
  return;
}


// Unmark events in line. We consider all marked events on line
// If such an event is inside a window openned from previous
// event from that line, we unmark is
//-------------------------------------------------------------------------
void
Task_packer::UnmarkEventsInLine( list<Unified_event> & events,
                          bool * is_event_marked)
{
  int max_izone_x = -1;

  // return if list_events is empty
  //-----------------------------------------------------------------------
  if ( events.empty() )    return;

  // iterators over list of events. it will point to event on coordinate
  // ((*it).x,y)
  //-----------------------------------------------------------------------
  unsigned int i_event     = 0;
  list<Unified_event>::iterator it       = events.begin();
  list<Unified_event>::iterator list_end = events.end();

  // If no event windows covering line y, Check if some events in list
  // are covered by previous events in list
  while ( it != list_end ) {
    // each event that is candidate to be a trigegr for openning a window
    // should participate
    if ( is_event_marked[i_event] ) {
      bool ignore = packer_helper.IgnoreInclusionZone(*it);
      if ( (!ignore) && ((*it).x < max_izone_x)  ) {
      // event *it will be inside inclusion zone of previous event
      is_event_marked[i_event] = false;
      }
      else {
      // A window will be openned around *it.
      // We update the max_izone_x so that we will unmark
      // subsequent events event if they are not inside overlapping windows
      // provided that heir x coordinate is smaller than max_izone_x.
      int ix0, iy0, ix1, iy1;
      packer_helper.InclusionZoneOfTrigger(*it,ix0,iy0,ix1,iy1);
      if ( ix1 > max_izone_x ) {
        max_izone_x = ix1;
      }
      }
    }

    ++it;
    ++i_event;
  }

  return;
}
```

```
// Create windows only around marked events
//-------------------------------------------------------------------------
bool
Task_packer::AddWindowsForMarkedEvents(list<Unified_event> & events,
                                       bool * is_event_marked,
                                       int y)
{
  int i_event = 0;
  e_windows->BeginUpdate();
  bool is_ok;
  for ( list<Unified_event>::iterator it = events.begin();
      it != events.end();
      ++it ) {
    if ( is_event_marked[i_event] ) {
      // Create a window and cels and features Sipdata subclasses
      Sipwinpack * winpack = NULL;
      if ( ds_trans ) {
      // Create winpack with the transformation that corresponds to line y
      // Pass onl2ref transformation to CreateWinpack

      // for top-down windows we would like to take transformation
      // corresponding to the lowest y coordinate rather that the
      // transformation of the center.
      if ( (*it).type == Unified_event::TOP_DOWN ) {
        // top down trigger
        const Sip_topdown & td
          = packer_helper.GetTdEvent((*it).top_down.index);
        int y_max = (*it).y + td.radius_y;
        if ( y_max > ds_trans->LargestLine() ) {
          mlog(LOG_STANDARD, "AddWindowsForMarkedEvents: "
             "y_max = %d but last line put in ds_trans = %d.",
             y_max,ds_trans->LargestLine() );
          y_max = ds_trans->LargestLine();
        }

        winpack =
          packer_helper.CreateWinpack( *it, ds_trans->Trans(y_max), client_id,
is_ok);
      }
      else {
        winpack =
          packer_helper.CreateWinpack( *it, ds_trans->Trans(y), client_id,
is_ok);
      }
      }
      else {
      // no transformation specified,
      // Pass self2aligned transformation to CreateWinpack
      camera2aligned.SetContextRange(CoordSys::Reference);
      winpack = packer_helper.CreateWinpack( *it, camera2aligned, client_id,
is_ok );
      }

      if ( winpack ) {
      // sipwinpack contains a non empty definition for a test window
      if ( !e_windows->PutWindow(winpack) ) {
        mlog(LOG_DERIVED_ERROR, "AddWindowsForMarkedEvents: "
             "Task %s :Failed to put window into system of windows"
```

```
            " from events defined on line %d.", Name());
        delete winpack;
        return false;
      }
      }
      else {
      // NULL winpack.
      if ( !is_ok ) {
        // failed to create Sipwinpack object.
        mlog(LOG_DERIVED_ERROR,"AddWindowsForMarkedEvents: ");
        return false;
      }
      }
    }
    ++i_event;
  }

  if ( !e_windows->EndUpdate() ) {
    mlog(LOG_APPLIC_ERROR, "AddWindowsForMarkedEvents: "
       "Task %s : Failed to integrate window into system of windows.",
       Name());
    return false;
  }

  return true;
}



// Pack all events in list inside all windows intersecting the intersect
// line number index_i_line
//-------------------------------------------------------------------------
void Task_packer::PackEvents(int index_i_line, list<Unified_event> & events )
{
  Intersect_line  * i_line    = e_windows->GetIntersectLine(index_i_line);

  // return if no event windows covering line y OR if list_events is empty
  if ( i_line->empty() || events.empty() ) {
    return;
  }

  // iterators over list of events. it will point to event on coordinate
  // ((*it).x,y)
  //-------------------------------------------------------------------------
  list<Unified_event>::iterator it       = events.begin();
  list<Unified_event>::iterator list_end = events.end();

  // Move to first Xendpoint. Xendpoints iterator will point at an Xendpoint
  // with coordinate i_line->X(). The list of overlapping windows will
  // contain all event windows overlapping the segment between i_line->X()
  // and the Xendpoint with smaller X coordinate
  //-------------------------------------------------------------------------
  i_line->Xbegin();

  // while not_end_of_Xendpoints of intersected event windows
  //-------------------------------------------------------------------------
  while ( !(i_line->Xend()) ) {
    // Check if there are overlapping event window.
```

```
    unsigned int osize = i_line->osize();
    while ( (*it).x < i_line->X() ) {
      //  current_event is inside a set of overlapping event_windows.
      for (unsigned int iosize = 0; iosize < osize; ++iosize) {
      ((Sipwinpack *)(i_line->GetOvelappingWindow(iosize)))->push_back(*it);
      }
      ++it;
      if ( !(it != list_end) ) {
      return;       // end of list of events
      }
    }

    // (*it).x >= i_line->X() (Possibly we passed past the last event
    // in list. Move to next Xendpoint.
    //-------------------------------------------------------------------
    i_line->Xnext();
  }
  return;
}



// Pack all events in list inside all windows intersecting the intersect
// line number index_i_line
//-------------------------------------------------------------------------
void Task_packer::PackCels(int y, int index_i_line, Ds_array<Cel> * ds_cels)
{
  if ( !ds_cels ) return;              // no CELs requested to be packed

  Ds_array_iter<Cel>     citer( ds_cels ); // iterator ovel line of CELs
  Cel                  * celement;         // points to a cel from citer

  Cel_event<Cel>         ce;             // contains the cel event to be packed
  ce.y = y;                           // y coordinate of packwed cel event

  Intersect_line  * i_line    = e_windows->GetIntersectLine(index_i_line);


  // return if no event windows covering line y OR if list_events is empty
  if ( i_line->empty() || citer.empty_line(y) )
  {
    return;
  }

  // iterators over cels is positionned at the beginning of the line
  //-------------------------------------------------------------------------
  citer.BeginLine(y);
  celement = citer.Next();
  if ( !celement )
  {
    return;             // no CELs in this line
  }


  // Move to first Xendpoint. Xendpoints iterator will point at an Xendpoint
  // with coordinate i_line->X(). The list of overlapping windows will
  // contain all event windows overlapping the segment between i_line->X()
```

```
  // and the Xendpoint with smaller X coordinate
  //-------------------------------------------------------------------------
  i_line->Xbegin();

  // while not_end_of_Xendpoints of intersected event windows and not end of
  // CELs
  //-------------------------------------------------------------------------
  while ( !(i_line->Xend()) )
  {
    // Check if there are overlapping event window.
    unsigned int osize = i_line->osize();
    while ( (int)(celement->x) < i_line->X() )
    {
        //    if ( celement->edge_code != 7 )
        //      {
      // pack cel only if do not contain skeleton data
      //  current_event is inside a set of overlapping event_windows.
      ce.x    = celement->x;
      ce.data = *celement;
      for (unsigned int iosize = 0; iosize < osize; ++iosize)
      {
        ((Sipwinpack *)(i_line->GetOvelappingWindow(iosize)))->push_back(ce);
      }
      //} // if ( celement->edge_code != 7 )
      celement = citer.Next(); // Move to next CEL
      if ( !celement )
      {
      return;           // No more CELs
      }
    }

    // celement->X() >= i_line->X() (Possibly we passed past the last cel)
    // Move to next Xendpoint.
    //-----------------------------------------------------------------------
    i_line->Xnext();
  }
  return;
}


// Pack all color cels in list inside all windows intersecting the intersect
// line number index_i_line
//-------------------------------------------------------------------------
void Task_packer::PackColorCels(int y,
                        int index_i_line,
                        Ds_array<Color_cel> * ds_color_cels)
{
  if ( !ds_color_cels ) return;     // no color CELs requested to be packed

  Ds_array_iter<Color_cel> citer( ds_color_cels ); //iterator ovel line of CELs
  Color_cel            * celement;         // points to a cel from citer

  Cel_event<Color_cel>         ce;  // contains the cel event to be packed
  ce.y = y;                  // y coordinate of packwed cel event

  Intersect_line  * i_line    = e_windows->GetIntersectLine(index_i_line);
```

```
  // return if no event windows covering line y OR if list_events is empty
  if ( i_line->empty() || citer.empty_line(y) )
  {
    return;
  }

  // iterators over cels is positionned at the beginning of the line
  //-------------------------------------------------------------------------
  citer.BeginLine(y);
  celement = citer.Next();
  if ( !celement )
  {
    return;              // no CELs in this line
  }


  // Move to first Xendpoint. Xendpoints iterator will point at an Xendpoint
  // with coordinate i_line->X(). The list of overlapping windows will
  // contain all event windows overlapping the segment between i_line->X()
  // and the Xendpoint with smaller X coordinate
  //-------------------------------------------------------------------------
  i_line->Xbegin();

  // while not_end_of_Xendpoints of intersected event windows and not end of
  // CELs
  //-------------------------------------------------------------------------
  while ( !(i_line->Xend()) )
  {
    // Check if there are overlapping event window.
    unsigned int osize = i_line->osize();
    while ( (int)(celement->x) < i_line->X() )
    {
      ce.x    = celement->x;
      ce.data = *celement;
      for (unsigned int iosize = 0; iosize < osize; ++iosize)
      {
      ((Sipwinpack *)(i_line->GetOvelappingWindow(iosize)))->push_back(ce);
      }
      celement = citer.Next(); // MOve to next CEL
      if ( !celement )
      {
      return;            // No more CELs
      }
    }

    // celement->X() >= i_line->X() (Possibly we passed past the last cel)
    // Move to next Xendpoint.
    //-----------------------------------------------------------------------
    i_line->Xnext();
  }
  return;
}


// Print CEL data (used for debugging only)
//-------------------------------------------------------------------
void Task_packer::PrintCelData(int y, Ds_array<Cel> * ds_cels)
{
```

```
  Ds_array_iter<Cel>     citer( ds_cels ); // iterator ovel line of CELs
  Cel                  * celement;         // points to a cel from citer
  int                    index;

  mlog(LOG_ALL, "PrintCelData: "
      "Line %d #CELs = %d [index=(x,edge_code)] -> ",y,citer.size_line(y));

  // return if no event windows covering line y OR if list_events is empty
  if ( citer.empty_line(y) )
  {
      mlog(LOG_ALL, "PrintCelData: "
      "NO CELS");
    return;
  }

  // iterators over cels is positionned at the beginning of the line
  //-------------------------------------------------------------------------
  citer.BeginLine(y);
  celement = citer.Next();
  if ( !celement )
  {
    mlog(LOG_ALL, "PrintCelData: "
        "ERROR");
    return;              // no CELs in this line
  }

  //-------------------------------------------------------------------------
  index = 0;
  while ( true )
  {
    mlog(LOG_ALL, "PrintCelData: "
        "%d=(%d,%d) ",index,celement->x,celement->edge_code);
    celement = citer.Next(); // MOve to next CEL
    if ( !celement )
    {
      mlog(LOG_ALL, "PrintCelData: "
          ". .. . DONE\n");
      return;            // No more CELs
    }
    ++index;
  }
}

/*
 *  $Log: task_packer.C,v $
 *  Revision 1.72  2001/02/20 10:09:01  sharond
 *  Allow masking inside a window (PIM in win)
 *
 *  Code Review by
 *
 *  Revision 1.71  2001/02/01 14:54:08  shmulik
 *  Adding printouts of number of elements (CELs, features, etc.) packed
 *  by packr task.
 *
 *  Code Review by
 *
 *  Revision 1.70  2000/09/06 05:40:19  meirm
 *  Producing ordered queues as output. Correcting
```

```
*  defficiency of e_windows in the index of last
*  line packed as reported.
*
*  Code Review by Meir
*
*  Revision 1.69  2000/05/24 11:11:02  shmulik
*  Improving robustness by ignoring triggers for window creation that are
*  too close to camera boundaries.
*
*  Code Review by
*
*  Revision 1.68  2000/02/21 12:38:28  shmulik
*  Adding client_id parameter, taken from camera_id and
*  sending it to CreateWinPack method.
*
*  Code Review by
*
*  Revision 1.67  1999/12/29 12:23:31  meirm
*  Code Review by
*
*  Revision 1.66  1999/12/23 08:57:53  shmulik
*  Changing return va;ue of Type() ifrom string into const char *
*
*  Code Review by Meir
*
*  Revision 1.65  1999/10/24 14:23:52  joseph-w
*  changing to derived error
*
*  Code Review by
*
*  Revision 1.64  1999/09/26 16:09:56  joseph-w
*  changing the order of including the x.H file to first line of x.C
*
*  Revision 1.63  1999/09/07 09:56:48  shmulik
*  Packing also SDD/COMRAD defects.
*
*  Revision 1.62  1999/08/29 11:52:48  rina
*  changed due to changes in Unified_event.
*
*  Revision 1.61  1999/08/18 09:52:14  shmulik
*  Major changes. Moving many methods to packer_helper class.
*  eliminating the use of events_services.
*
*  Revision 1.60  1999/07/19 09:58:05  shmulik
*  Adding guard for preventing the dereferencing of NULL pointers
*  in case of empty (input) ds_trans.
*
*  Revision 1.59  1999/07/12 13:39:18  sharond
*  Cosmetics changes
*
*  Revision 1.58  1999/07/07 12:56:57  sharond
*  New coordinate model
*
*  Revision 1.57  1999/06/22 17:08:05  shmulik
*  Using correctly new coordinate systems of base class. Storing localy
*  needed camera2align and align2camera transformations. Checking that
*  all used data sources are camera data sources
*
```

```
*  Revision 1.56  1999/06/13 16:43:07  shmulik
*  Replacing Max_Line_Size method with MaxLargestPixelInScan
*
*  Revision 1.55  1999/06/08 08:11:22  sharond
*  Strict Context checking in CoordSys
*
*  Revision 1.54  1999/03/01 13:48:49  sharond
*  Arrange logging messages
*
*  Revision 1.53  1999/02/28 18:42:33  shmulik
*  Changing transformation warning line 1760 to STANDARD
*
*  Revision 1.52  1999/02/28 08:17:15  joseph-w
*  shared memory enable/disable issues
*
*  Revision 1.51  1999/02/10 14:38:16  joseph-w
*  minor language improvments
*
*  Revision 1.50  1999/01/19 15:05:25  shmulik
*  Removing inline from virtual methods
*
*  Revision 1.49  1998/12/24 07:57:23  sharond
*  Enable multi slice cel reference & extraction
*
*  Revision 1.48  1998/12/16 14:41:57  sharond
*  Check Success / Failure of TransOp operations (Mult&Div)
*
*  Revision 1.47  1998/12/13 07:21:33  shmulik
*  Insure changes
*
*  Revision 1.46  1998/11/25 11:42:22  joseph-w
*  added missing return code
*
*  Revision 1.45  1998/11/25 07:03:34  shmulik
*  changing methods returning void into methods returning bool.
*
*  Revision 1.44  1998/11/05 15:52:37  sharond
*  after demo effect
*
*  Revision 1.43  1998/11/04 14:15:53  joseph-w
*  more compliance to C++ standards
*
*  Revision 1.42  1998/11/01 15:13:12  joseph-w
*  *** empty log message ***
*
*  Revision 1.41  1998/10/26 13:48:55  joseph-w
*  KCC & egcs compatibility issues
*
*  Revision 1.40  1998/10/16 15:47:57  sharond
*  *** empty log message ***
*
*  Revision 1.39  1998/10/15 17:27:29  shmulik
*  *** empty log message ***
*
*  Revision 1.38  1998/10/14 15:19:44  shmulik
*  minor changes
*
*  Revision 1.37  1998/10/14 09:30:13  sharond
```

```
*  *** empty log message ***
*
*  Revision 1.36  1998/10/08 07:32:14  sharond
*  *** empty log message ***
*
*  Revision 1.35  1998/10/06 14:04:07  sharond
*  Multi slice Balls handling
*
*  Revision 1.34  1998/09/28 18:34:47  shmulik
*  arrangement of code
*
*  Revision 1.33  1998/09/28 17:28:14  shmulik
*  Tranfroming fixed top down widnows beforing creating
*  Sipwinpack. changing parameter to CreateSipwinpack from
*  ref2onl into onl2ref.
*
*  Revision 1.32  1998/09/27 19:19:31  shmulik
*  Adding interface to help defining windows with
*  a proper coordinate system.
*
*  Revision 1.31  1998/09/23 15:00:22  shmulik
*  Adding function element to window's private members
*  and removing it from sipwinpack. making Win_queue
*  to contain pointers to Sipwin instead of pointers to
*  Sipeinpack.
*
*  Revision 1.30  1998/08/25 07:52:24  sharond
*  *** empty log message ***
*
*  Revision 1.29  1998/08/23 17:26:48  joseph-w
*  fixed warnings
*
*  Revision 1.28  1998/08/11 12:27:37  sharond
*  Camera Model + Sip General Data modifications
*
*  Revision 1.27  1998/07/19 17:15:11  shmulik
*  Minor changes
*
*  Revision 1.26  1998/05/21 11:35:21  shmulik
*  Improving code for unification of events (minimizing
*  number of openned windows)
*
*  Revision 1.25  1998/05/19 17:41:39  shmulik
*  Adding data persistance to data sources and the
*  concept of sync Vs. non_sync data sources used by tasks
*
*  Revision 1.24  1998/05/11 17:21:24  shmulik
*  changes for allowing to pack skeletons inside windows
*
*  Revision 1.23  1998/05/05 06:24:06  shmulik
*  code improvements
*
*  Revision 1.22  1998/04/08 07:09:33  eyalk
*  converting to string, to ansi conventions, new factory, etc ...
*
*  Revision 1.21  1998/02/19 07:24:54  eyalk
*  updates of new compiler
*
```

```
 *  Revision 1.20  1998/01/15 10:27:45  shmulik
 *  *** empty log message ***
 *
 *  Revision 1.19  1998/01/06 14:11:52  shmulik
 *  Replacing all LOG_CONFIG log messages by LOG_APPLIC
 *
 *  Revision 1.18  1998/01/01 09:59:05  shmulik
 *  Integrating new logger into the SIP
 *
 *  Revision 1.17  1997/12/29 07:46:32  yulia
 *  Adding NEW logger
 *
 *  Revision 1.16  1997/12/15 12:57:32  shmulik
 *  Adding support for color CELs
 *
 *  Revision 1.15  1997/12/14 14:30:16  shmulik
 *  adding support for templated version of CEL data structures
 *
 *  Revision 1.14  1997/10/12 07:55:01  shmulik
 *  *** empty mlog message ***
 *
 *  Revision 1.13  1997/09/25 06:56:17  shmulik
 *  adding general_data variable to func DoSetParams
 *
 *  Revision 1.12  1997/06/22 06:26:54  shmulik
 *  *** empty mlog message ***
 *
 *  Revision 1.11  1997/06/18 06:38:04  shmulik
 *  bug fixes
 *
 *  Revision 1.10  1997/05/28 11:19:17  shmulik
 *  MOdifications
 *
 *  Revision 1.9  1997/05/20 17:56:26  shmulik
 *  Purification and bug fixes
 *
 *  Revision 1.8  1997/05/01 13:17:19  shmulik
 *  bug fix
 *
 *  Revision 1.7  1997/05/01 11:22:22  shmulik
 *  Improved handling of transformation of top down reference to online lines
 *
 *  Revision 1.6  1997/04/15 17:37:03  shmulik
 *  nothing much
 *
 *  Revision 1.5  1997/04/15 17:18:17  eyalk
 *  Incorperating changes in basic_cel
 *
 *  Revision 1.4  1997/04/06 14:47:25  shmulik
 *  Adding ability for ignoring inclusion zones. A window will be created aroung
triggers inside inclusion zone of other windows if the ignore_izone_flag is set
 *
 *  Revision 1.3  1997/03/31 10:18:58  shmulik
 *  minor improvements of code
 *
 *  Revision 1.2  1997/03/18 15:56:09  shmulik
 *  Making things compatible to data arriving from hardware
 *
```

```
 *  Revision 1.1  1997/02/18 14:42:11  shmulik
 *  *** empty mlog message ***
 *
 *
 */


/*
//--------------------------------------------------------------------
// Compute needed top down lines. We need to have all top down events in
// lines (on line coordinates) [td_first_line,td_last_line].
//
// The required top down lines are stored in onl_td_events where
// onl_td_events[y] is the list (sorted by X coordinates) of all top down
// events defined for line y in the online coordinate system.
//
// Note that
// ALL top down events are originally defined in reference coordinates.
//
// The computation of the range of top-down lines needed is based on the
// demand for top down lines during the computation of packer (see methods
// ProcessEventsInLine and ProcessEventsInFirstLine).
//--------------------------------------------------------------------
void Task_packer::ComputeNeededTopDownLines(int first_line, int last_line )
{
  // compute td_first_line (first needed top down line) and td_last_line
  // (last needed top down line)
  //-------------------------------------------------------------------
  ComputeRangeOfNeededTopDownLines(first_line,last_line,
                           td_first_line,td_last_line);
  if ( td_first_line < 0 ) return;

  // A transformation bteween online and reference coordinate systems is
  // defined
  //
  // devide the range [td_first_line,td_last_line] into segments
  // [first_onl_line,last_onl_line], where the first first_onl_line is equal
  // to td_first_line and the last_onl_line is equal to td_last_line.
  //
  // The segments are such that the transformation between on line and
  // reference coordinate systems is the same for each segment.
  //-------------------------------------------------------------------
  int first_onl_line = td_first_line;
  while ( true )
  {
    // compute last endpoint of segment beginning with first_onl_line
    int last_onl_line;
    if ( ds_trans ) {
      last_onl_line = ds_trans->MaxLineOfInfluence(first_onl_line);
      if ( last_onl_line > td_last_line ) last_onl_line = td_last_line;
    }
    else {
      last_onl_line = td_last_line;
    }

    // get range of reference lines covered by the rectangle
    // [0,Sip_general_data::MaxLargestPixelInLine_cameras()]
    //      X
```

```
    // [first_corrected_onl_line,last_onl_line]
    // where first_corrected_onl_line
    //           = first_onl_line - RegistrationRangeOverlap
    // (see events_services.H). The correction with
    // RegistrationDeviationTolerance is performed because we want to cover
    // all lines in the reference so that we shall not loose any top down
    // event.
    int yref_min, yref_max;
    int first_corrected_onl_line =
      first_onl_line - ds_events_services->RegistrationRangeOverlap();
    if ( first_corrected_onl_line < 0 ) first_corrected_onl_line = 0;

    ComputeBoundingLines(0, first_corrected_onl_line,
                  Sip_general_data::Largest_pixel_in_line(
UsedCoordinate_System() ),
                  last_onl_line,
                  yref_min,
                  yref_max);

    // Get the top down events (after transformation to online coordinates)
    // for all reference lines covered by the above rectangle.
    for ( int yref = yref_min; yref <= yref_max; ++yref )
    {
      GetTopDownEventsFromReferenceLine(yref,
                            first_corrected_onl_line,
                            first_onl_line,
                            last_onl_line);
    }

    // move to next segment (or break at last segment)
    if ( last_onl_line == td_last_line ) break;
    first_onl_line = last_onl_line + 1;
  }

  // sort all online topdown events by their X coordinates
  //-------------------------------------------------------------------------
  for ( int yonl = td_first_line; yonl <= td_last_line; ++yonl )
  {
    onl_td_events[yonl].sort();
  }

  return;
}

//-------------------------------------------------------------------------
// Compute range of needed top down lines.
// The computation of the range of top-down lines needed is based on the
// demand for top down lines during the computation of packer (see methods
// ProcessEventsInLine and ProcessEventsInFirstLine).
// All coordinates (input and output) are in the online coordinate system.
//-------------------------------------------------------------------------
void Task_packer::ComputeRangeOfNeededTopDownLines(int   first_line,
                                        int   last_line,
                                        int & td_first_line,
                                        int & td_last_line)
{
  int td_priority  = packer_helper.td_priority;
  int td_offset    = max_window_size * ( td_priority - min_priority + 1);
```

```
  int largest_line = Sip_general_data::Largest_line_in_scan(
UsedCoordinate_System() );

  // compute td_first_line (first needed top down line) and td_last_line
  // (last needed top down line)
  //-------------------------------------------------------------------------
  if ( first_line == 0 )
  {
    td_first_line = 0;
  }
  else
  {
    td_first_line = first_line + td_offset;
  }

  if ( td_first_line > largest_line )
  {
    // in this case, we will not use top down lines
    td_first_line = -1;
    td_last_line  = -1;
    return;
  }

  // compute last_line needed top down line
  td_last_line = last_line + td_offset;
  if ( td_last_line > largest_line )
  {
    td_last_line = largest_line;
  }

  return;
}

//-------------------------------------------------------------------------
// Compute bounding lines (in reference coordinate system) of a given
// rectangle (given in online coordinate system). It is assumed that the
// online-->reference transformation is the same for all lines between yonl_min
// and yonl_max.
//-------------------------------------------------------------------------
void Task_packer::ComputeBoundingLines(int xonl_min, int yonl_min,
                                       int xonl_max, int yonl_max,
                                       int & yref_min,
                                       int & yref_max)
{
  // get transformation for the range of lines between line f and line l
  // (including f and l)
  const Affine2dtrans & onl2ref = (ds_trans ? ds_trans->Trans(yonl_min) :
camera2aligned);

  // compute the image of the rectangle
  // [xonl_min,xonl_max] X [yonl_min,yonl_max]
  // on the reference coordinate system. The rectangle is transformed
  // into the quadrilateral with vertices (xref[i],yref[i]), i = 0,1,2,3
  double xref[4], yref[4];

  onl2ref(xref[0], yref[0], xonl_min, yonl_min );
  onl2ref(xref[1], yref[1], xonl_min, yonl_max );
  onl2ref(xref[2], yref[2], xonl_max, yonl_max );
```

```
  onl2ref(xref[3], yref[3], xonl_max, yonl_min );

  // compute bounding lines of reference rectangle: ymin and ymax
  double yd_min = yref[0];
  double yd_max = yref[0];
  for ( int i = 1; i < 4; ++i )
  {
    if ( yref[i] < yd_min ) yd_min = yref[i];
    if ( yref[i] > yd_max ) yd_max = yref[i];
  }

  // compute min and max line so that they are within the range of ref lines
  // [0,NFTDuevents.size()]
  yref_min  = (int)(floor(yd_min));
  if ( yref_min < 0 ) yref_min = 0;

  Sip_lines_of_uevents & NFTDuevents = packer_helper.NFTDUevents();
  yref_max  = (int)(ceil(yd_max));

  if ( yref_max >= (int)NFTDuevents.size() )
  {
    yref_max = NFTDuevents.size() - 1;
  }

  return;
}

//-------------------------------------------------------------------------
// Get top down events in online coordinate system from all reference
// top down events defined on line y (in ref coordinate system).
// Process only top down events defined for lines between first_onl_line
// and last_onl_line (where the transformation is the same).
//-------------------------------------------------------------------------
void
Task_packer::GetTopDownEventsFromReferenceLine(int yref,
                                    int first_corrected_onl_line,
                                    int first_onl_line,
                                    int last_onl_line )
{
  // Get the inverse of the (fixed) transformation defined for all online
  // lines between first_onl_line and last_onl_line (both ends included)
  const Affine2dtrans & ref2onl =
    ( ds_trans ? ds_trans->InverseTrans(first_onl_line) : aligned2camera );

  double dfirst_onl_line = (double)first_corrected_onl_line;
  double dlast_onl_line  = (double)last_onl_line;

  // run over all TD events defined on line y (ref coordinate system)
  Sip_lines_of_uevents & NFTDuevents = packer_helper.NFTDUevents();
  list<Unified_event> & ref_td       = NFTDuevents[yref];
  list<Unified_event>::iterator it   = ref_td.begin();
  while ( it != ref_td.end() )
  {
    // transform from ref coordinate system to online coordinate system

    // reasonable defaults:
    double xd_onl(0.0), yd_onl(0.0);
    // set xd_onl, yd_onl
```

```
    ref2onl( xd_onl, yd_onl, (*it).x, (*it).y  );
    // check
    if ( (yd_onl >= dfirst_onl_line) && (yd_onl <= dlast_onl_line) )
    {
      // inside range of required online lines.
      list<Unified_event>::iterator cit = it;
      ++it;

      // transform coordinate xd_onl into subpixel representation
      double dint                = floor(xd_onl + 0.5); // changed from rint() -
GNU specific
      (*cit).x                   = (int)dint;
      (*cit).top_down.x_sub_pixel= (int)((xd_onl-dint)*CEL_SUB_PIXEL_BITS);

      // transform coordinate yd_onl into subpixel representation
      dint                       = floor(yd_onl + 0.5);
      int    yi_onl              = (int)dint;
      (*cit).y                   = yi_onl;
      (*cit).top_down.y_sub_pixel= (int)((yd_onl-dint)*CEL_SUB_PIXEL_BITS);

      // remove TD event *cit from list ref_td into the list
      // onl_td_events[yi_onl].
      if ( yi_onl < first_onl_line )
      {
      // Note that yi_onl might be smaller
      // than first_onl_line. In this case we store the corresponding
      // TD event on the list containing events for line first_onl_line.
      yi_onl = first_onl_line;
      }
      onl_td_events[yi_onl].splice(onl_td_events[yi_onl].end(),ref_td,cit);
    }
    else
    {
      // not inside range of required online lines.
      ++it;
    }
  }

  return;
}


//---------------------------------------------------------------------------
// Create all fixed windows. The size and location of those windows are
// known before scan.
//---------------------------------------------------------------------------
bool Task_packer::CreateFixedWindows( int last_line )
{
  // get the vector of top down objects
  bool is_ok;
  Sip_topdown_vector & v_topdown = ds_events_services->TopDownVector();

  e_windows->BeginUpdate();
  Unified_event td;
  for ( unsigned int i = 0; i < v_topdown.size(); ++i )
  {
    if ( v_topdown[i].is_fixed )
    {
```

```
Sipwinpack * winpack = NULL;

// td is the top down event trigering the creation of this event.
td.x              = v_topdown[i].x_center;
td.y              = v_topdown[i].y_center;
td.type           = Sip_event_header::TOP_DOWN;
td.top_down.index = i;

if ( ds_trans )
{
// Create winpack with the transformation that corresponds to the first
// line in the scan
if ( td.y < last_line )
{
  // td.y is a line within the range of lines covered by registration
  // transformation. Pass onl2ref transformation to CreateWinpack
  const Affine2dtrans & onl2ref = ds_trans->Trans(td.y);
  const Affine2dtrans & ref2onl = ds_trans->InverseTrans(td.y);
  td.Transform(ref2onl);
  winpack =
    ds_events_services->CreateWinpack( td, onl2ref,  is_ok );
}
else
{
  // td.y is a line greater than the max line for which
  // transformation was computed. Use the last transformation
  // computed so far. Pass onl2ref transformation to CreateWinpack.
  const Affine2dtrans & onl2ref = ds_trans->Trans(last_line);
  const Affine2dtrans & ref2onl = ds_trans->InverseTrans(last_line);
  td.Transform(ref2onl);
  winpack =
    ds_events_services->CreateWinpack(td, onl2ref, is_ok );

}
}
else
{
// no transformation specified,
// Pass aligned2ref transformation to CreateWinpack
td.Transform(aligned2camera);
winpack = ds_events_services->CreateWinpack( td, camera2aligned, is_ok );
}
mlog(LOG_ALL, "CreateFixedWindows: "
   "Topdown trigger(%d %d) transfored into trigger (%d %d)  \n",
   v_topdown[i].x_center,v_topdown[i].y_center,
   td.x, td.y);

if ( winpack )
{
// sipwinpack contains a non empty definition for a test window
if ( !e_windows->PutWindow(winpack) )
{
  mlog(LOG_APPLIC_ERROR, "CreateFixedWindows: "
       "Task %s :Failed to put fixed window into system of "
       "windows.", Name());
  delete winpack;
  return false;
}
```

```
        }
        else
        {
        // NULL winpack.
        if ( !is_ok )
        {
          // failed to create Sipwinpack object.
          mlog(LOG_DERIVED_ERROR,"CreateFixedWindows: ");
          return false;
        }
        }
      } // if ( v_topdown[i].is_fixed )
    } // end of for loof

    e_windows->EndUpdate();

    return true;
}
*/
```

```
#include "task_test_manager.H"

#include <string>
#include <iterator>
using namespace std;

#include "sipdata_sub_windows.H"
#include "id_singleton.H"
#include "sip_logger.H"
#include "sip_config.H"
#include "sipwin.H"
#include "set_operation.H"
#include "sip_general_data.H"
#include "win_queue.H"

// add a factory for Task_data_transfer to list of factories
//-----------------------------------------------------------------------
static Register_subclass<Task_test_manager> r_Task_test_manager;

Task_test_manager::Task_test_manager()
  : ds_queue(NULL)
#ifdef MP_SAFE
, thrm(*this), Sub_test_manager_started(false)
#endif // MP_SAFE
{
  buffer.reserve(MAX_WINDOWS_PER_THREAD);
}

Task_test_manager::~Task_test_manager()
{ }

const char * Task_test_manager::Type() const
{
  return "Task_test_manager";
}

Base_factory *   Task_test_manager::DoCreate() const
{
  return new Task_test_manager;
}


// virtual base class functions are those of the RootTask
bool Task_test_manager::DoLoadConfiguration( Sip_config & config )
{
  // locate Queue source
  //-----------------------------------------------------------------
  if ( v_ds_used_sync.size() == 1 ) {
    ds_queue = dynamic_cast<Win_queue *>(v_ds_used_sync[0]);
    if (!ds_queue) {
      mlog(LOG_APPLIC_ERROR, "DoLoadConfiguration: "
         "Task %s should have a Win_queue data source ",Name());
      return false;
    }
  } else {
    mlog(LOG_APPLIC_ERROR, "DoLoadConfiguration: "
       "Task %s should have a single data source ",Name());
    return false;
```

```
  }


  // Search for all output queues.
  vect_out_queues.clear();
  vect_out_queues_name_ref.clear();
  for ( unsigned int i = 0; i < v_ds_produced.size(); ++i ) {
    Win_queue * pp = dynamic_cast<Win_queue *>(v_ds_produced[i]);
    if ( pp != NULL ) {
      // Put queue ptr to first vector and integer reference to name of queue to
      // second vector.
      vect_out_queues.push_back(pp);
      vect_out_queues_name_ref.push_back(Id_singleton::GetId( string(pp->Name())
) );
    } else {
      // i'th data source is not a Win_queue data source
      mlog(LOG_APPLIC_ERROR, "DoLoadConfiguration: "
         "Task %s: The %d'th data source produced "
         "is not a Win_queue data source", Name());
      DoClear();
      return false;
    }
  }

  // locate affine transformation source
  //-----------------------------------------------------------------------
  if ( v_ds_used_non_sync.size() > 0  ) {
    mlog(LOG_APPLIC_ERROR,
       "Task %s can not have "
       "non synchronous data source ",Name());
    DoClear();
    return false;
  }

  if ( !config.GetFieldElement("number_of_threads",number_of_threads) ) {
    mlog(LOG_DERIVED_ERROR, "DoLoadConfiguration: "
       "Failed to get number_of_threads field (Task %s) "
       "from configuration",
       Name());
    DoClear();
    return false;
  }

  if (number_of_threads > 7) {
    mlog(LOG_APPLIC_ERROR,"DoLoadConfiguration: number_of_threads provided (%d)
is "
       "above the current limit of 7 (task %s)",number_of_threads,Name());
    DoClear();
    return false;
  }

  Sub_test_manager_started = false;
  // set according to if threading is disabled or not
  minimum_windows_per_thread = (number_of_threads != 0) ? MIN_WINDOWS_PER_THREAD
: 999999;

  return true;
}
```

```
// DoClear: Undo DoLoadConfiguration. This function must work even if
// called before DoLoadConfiguration or after error in DoLoadConfiguration
//-----------------------------------------------------------------------
bool Task_test_manager::DoClear()
{
  ds_queue = NULL;
  vect_out_queues.clear();

#ifdef MP_SAFE
  // Shut down threads
  if (Sub_test_manager_started) {
    Sub_test_manager_started = false;
    if (!thrm.user_close() ) {
      mlog(LOG_DERIVED_ERROR,"DoClear");
      return false;
    }
  }
#endif // MP_SAFE

  return true;
}


// Do InitScan
//-----------------------------------------------------------------------
bool Task_test_manager::DoInitScan()
{
#ifdef MP_SAFE
  // start up the threads
  if ( (!Sub_test_manager_started) && (number_of_threads > 0) ) {
    if (!thrm.user_open(number_of_threads,MAX_WINDOWS_PER_THREAD) ) {
      mlog(LOG_DERIVED_ERROR,"DoInitScan");
      return false;
    }
    Sub_test_manager_started = true;
  }
#endif //MP_SAFE

  return true;
}


// undo InitScan
//-----------------------------------------------------------------------
bool Task_test_manager::DoUnInitScan()
{
  return true;
}


// Do process windows
//-----------------------------------------------------------------------
bool Task_test_manager::DoProcess( const int n_units )
{
  int n_to_work = n_units;
  // CHECK for availablity of data
```

```
  if (ds_queue->size() < n_units) {
    n_to_work = ds_queue->size();
    if (n_to_work == -1) {
      mlog(LOG_DERIVED_ERROR,"DoProcess");
      return false;
    }
    mlog(LOG_WARN, "DoProcess: asked to work on %d windows "
       "but found only %u in queue %s.",n_units,n_to_work,
       ds_queue->Name());
  }
  // check for min data for multi-threading
  if ((n_to_work / (number_of_threads+1)) < minimum_windows_per_thread) {
    // no multi - thread
    if (!ds_queue->Pop_n(back_insert_iterator<vector<Sipwin*> >(buffer),
                  n_to_work)) {
      mlog(LOG_DERIVED_ERROR,"DoProcess");
      // erase windows
      while (!buffer.empty()) {
      Sipwin * w = buffer.back();
      buffer.pop_back();
      if (w) {
        delete w;
      }
      }
      return false;
    }
    if (!SubProcess(buffer)) {
      mlog(LOG_DERIVED_ERROR,"DoProcess");
      // erase windows
      while (!buffer.empty()) {
      Sipwin * w = buffer.back();
      buffer.pop_back();
      if (w) {
        delete w;
      }
      }
      return false;
    }
    // DONE!
  } else { // multi-thread
#ifdef MP_SAFE
    unsigned int windows_per_thread = n_to_work / (number_of_threads+1);
    // get windows for worker threads.
    for (unsigned int i = 0; i < number_of_threads; ++i) {
      if (!ds_queue->Pop_n(back_insert_iterator<vector<Sipwin*>
>(thrm.win_queues[i]),
                     windows_per_thread)) {
      mlog(LOG_DERIVED_ERROR,"DoProcess");
      // kill windows.
      for (unsigned int j = 0; j <= i; ++j) {
        vector<Sipwin*> & local_buffer = thrm.win_queues[j];
        while (!local_buffer.empty()) {
          Sipwin * w = local_buffer.back();
          local_buffer.pop_back();
          if (w) {
            delete w;
          }
        }
```

```
      }
      return false;
      }
    }
    // set the threads loose ... any exit must now wait for threads to end.
    thrm.wall->wait();
    // get windows for working in this thread - this will include any leftovers
    // from rounding the above division
    if (!ds_queue->Pop_n(back_insert_iterator<vector<Sipwin*> >(buffer),
                  n_to_work - (windows_per_thread * number_of_threads))) {
      mlog(LOG_DERIVED_ERROR,"DoProcess");
      // erase windows
      while (!buffer.empty()) {
      Sipwin * w = buffer.back();
      buffer.pop_back();
      if (w) {
        delete w;
      }
      }
      // we gotta wait for the threads now.
      thrm.wall->wait();
      // don't care about thread results
      return false;
    }
    // work on local windows
    if (!SubProcess(buffer)) {
      mlog(LOG_DERIVED_ERROR,"DoProcess");
      // erase windows
      while (!buffer.empty()) {
      Sipwin * w = buffer.back();
      buffer.pop_back();
      if (w) {
        delete w;
      }
      }
      // wait for workers to finish
      thrm.wall->wait();
      // exit
      return false;
    }
    // synch against worker threads
    thrm.wall->wait();
    // done multi-thread.
#endif // MP_SAFE
  }
  return true;
}

bool Task_test_manager::SubProcess ( vector<Sipwin*> & wins)
{
  // loop till queue is empty
  while (wins.size() != 0) {
    // get window
    Sipwin * w = wins.back();
    wins.pop_back();
    // check if null pointer
    if ( !w ) {
      mlog(LOG_APPLIC_ERROR, "SubProcess: "
```

```
      "Task %s: An empty window poped from queue.", Name());
   return false;
}
Sipwin & win = *w;
mlog(LOG_ALL, "SubProcess: "
    "Task %s: Run now window %s id(%d) [%d %d]x[%d %d].",
    Name(), win.Type(), win.Id(), win.X0(), win.X1(),
    win.Y0(), win.Y1() );

// ignore dummy EOD wakeup windows

if ( (win.Trigger()).type == Unified_event::END_OF_DATA ) {
  mlog(LOG_STANDARD,"SubProcess: "
      "End of scan detected by task %s.",Name());
  delete w;
  continue;
}

// execute function
Sipwinfunc::Return_code r_code = win.Run();

switch ( r_code ) {
case Sipwinfunc::ERROR:
  // take care of error conditions
  mlog(LOG_DERIVED_ERROR,"SubProcess: ");
  delete w;
  return false;
case Sipwinfunc::OK:
case Sipwinfunc::IGNORE:
  // end of handling of this window.
  mlog(LOG_ALL, "SubProcess: "
      "Task %s: EndOfHandling - deleting window.",Name());
  delete w;
  break;
case Sipwinfunc::FORWARD_THIS:
  // Transfer window to a forward destination.
  // transfer owwnership to SendToForwardQueue
  if ( !SendToForwardQueue(w) ) {
  mlog(LOG_DERIVED_ERROR,"SubProcess: ");
  return false;
  }
  break;
case Sipwinfunc::FORWARD_ALL: {
  if ( !ForwardSubWindows(win) ) {
  mlog(LOG_DERIVED_ERROR,"SubProcess: ");
  delete w;
  return false;
  }

  // Transfer also parent window to a forward destination.
  // transfer owwnership to SendToForwardQueue
  if ( !SendToForwardQueue(w) ) {
  mlog(LOG_DERIVED_ERROR,"SubProcess: ");
  return false;
  }
  break;
}
case Sipwinfunc::FORWARD_SUB_WINDOWS: {
```

```
      // withdraw and forward all subwindows
      if ( !ForwardSubWindows(win) ) {
      mlog(LOG_DERIVED_ERROR,"SubProcess: ");
      delete w;
      return false;
      }
      // delete parent window.
      delete w;
      break;
    }
    case Sipwinfunc::TRUE:
    case Sipwinfunc::FALSE:
      mlog(LOG_APPLIC_ERROR, "SubProcess: "
         "Task %s: TRUE/FALSE Return_codes for predicate function are not
expected here.",
         Name());
      delete w;
      return false;
    default:
      mlog(LOG_APPLIC_ERROR, "SubProcess: "
         "Task %s: Unrecognized return status.",Name());
      delete w;
      return false;
    }
  }
  // done
  return true;
}

// Withdraw Sipdata_sub_windows from win and try to forward them to
// output destination.
// In any case the Sipdata_sub_windows is deleted.
//-------------------------------------------------------------------------
bool Task_test_manager::ForwardSubWindows( Sipwin & win )
{
  // withdraw and forward all subwindows
  Sipdata_sub_windows * sub_windows =
    WIN_WITHDRAW(Sipdata_sub_windows,win);
  if ( !sub_windows ) {
    mlog(LOG_APPLIC_ERROR, "ForwardSubWindows: "
       "Task %s: Can not extract subwindows from window %s.",
       Name(),win.Type());
    return false;
  }

  for ( unsigned int i = 0; i < sub_windows->size(); ++i ) {
    // Transfer window to a forward destination.
    // transfer ownership of w_sub to SendToForwardQueue
    Sipwin * w_sub = sub_windows->WithdrawSubWin(i);
    if (w_sub) {
      if ( !SendToForwardQueue(w_sub) ) {
      mlog(LOG_DERIVED_ERROR,"ForwardSubWindows: ");
      delete sub_windows;
      return false;
      }
    }
  }
```

```
  delete sub_windows;
  return true;
}


// Try to forward window to output destination.
// If succeeded, window is pushed to output queue. Else window is
// deleted. In any case ownership of window is taaken.
//-----------------------------------------------------------------------
bool
Task_test_manager::SendToForwardQueue( Sipwin * w )
{
  // window transferred to a forward destination
  unsigned int i = 0;
  while (  (i < vect_out_queues.size()) ) {
    if ( vect_out_queues_name_ref[i] == (w->ForwardDestination()) ) {
      // found a destination queue
      mlog(LOG_ALL, "SendToForwardQueue: "
         "Task %s : Forwarding window to queue %s.",
         Name(),vect_out_queues[i]->Name());
#ifdef MP_SAFE
      // gotta protect all output queues
      if (number_of_threads != 0) thrm.mutex->acquire();
      vect_out_queues[i]->Push(w);
      if (number_of_threads != 0) thrm.mutex->release();
#else  // NOT MP_SAFE
      vect_out_queues[i]->Push(w);
#endif // MP_SAFE
      return true;
    }
    else {
      // found a destination queue
      mlog(LOG_ALL, "SendToForwardQueue: "
         "Task %s : Output queue %d (name == %s) is equal to forward destination
(%d)."
         "Detected for window %s.",
         Name(),
         vect_out_queues_name_ref[i],
         vect_out_queues[i]->Name(),
         (w->ForwardDestination()),
         w->Type());
    }
    ++i;
  }

  mlog(LOG_APPLIC_ERROR,
       "Task %s : Forward destination %d (%s) of window %s does not"
       " match any name of produced queues.",
       Name(),
       (w->ForwardDestination()),(Id_singleton::Id2String(w-
>ForwardDestination())).c_str(),
       w->Type());
  mlog(LOG_APPLIC_ERROR, "The window is:");
  w->Print();
  delete w;
  return false;
}
```

```
// Handle all side effects of task during scan:
// free allocations made during scan close files created during scan, etc.
//------------------------------------------------------------------------
bool Task_test_manager::DoEndScan()
{
  return true;
}

#ifdef MP_SAFE

// CTOR - Init ACE_Task with link to local ACE_Thread_Manager
Task_test_manager::
Sub_test_manager::Sub_test_manager(Task_test_manager & own)
  : // provide ACE_Task with our private ACE_Thread_Manager
  // although we provide a pointer to an unconstructed object
  // it will work as ACE_Task ctor just saves this pointer for
  // future use
  ACE_Task<ACE_MT_SYNCH>(&manager),
  wall(NULL), mutex(NULL), manager(), shutdown(false),
  n_threads(0), n_elements(0),
  owner(own)
{ }

// start up the service.
bool Task_test_manager::Sub_test_manager::
user_open(const unsigned int n_threads,
        const unsigned int n_elements_per_queue)
{
  // sanity check : 1 to 7 threads allowed
  if ( (n_threads < 1) || (n_threads > 7) ) {
    mlog(LOG_APPLIC_ERROR,"Sub_test_manager::user_open: got illegal number"
       " of threads (%u).",n_threads);
    return false;
  }
  // set n_elements; no check as this is used only for reserve()
  n_elements = n_elements_per_queue;
  // allocate barrier object. We wait for n_threads + ourselves
  wall = new ACE_Thread_Barrier(n_threads + 1);
  // allocate thread creation mutex
  mutex = new ACE_Thread_Mutex();
  if ( (!wall) || (!mutex)) {
    mlog(LOG_ALLOC_ERROR,"Sub_test_manager::user_open");
    if (wall) delete wall;
    if (mutex) delete mutex;
    return false;
  }
  // clear the shutdown flag
  shutdown = false;
  // thread creation
  if (activate(THR_NEW_LWP | THR_JOINABLE, n_threads) != 0) {
    mlog(LOG_APPLIC_ERROR,"Sub_test_manager::user_open: thread "
       "creation error");
    if (wall) delete wall;
    if (mutex) delete mutex;
    return false;
  }
  // wait for all threads to starup.
  wall->wait();
```

```
  // All threads are up and setup
  // Done ... threads should be now resting against wall barrier.
  return true;
}

// start up the service - nothing to do here
int Task_test_manager::Sub_test_manager::open(void *)
{
  mlog(LOG_ALL,"Sub_test_manager::open : starting up threads");
  return 0;
}

// nothing to do here
int Task_test_manager::Sub_test_manager::close(u_long)
{
  mlog(LOG_ALL,"Sub_test_manager::close : shutting down threads");
  return 0;
}

bool Task_test_manager::Sub_test_manager::user_close()
{
  // ignore call if wall doesn't exist - means that it has already been called.
  if (!wall) {
    mlog(LOG_WARN,"Sub_test_manager::user_close: called with "
       "no threads running.");
    return false;
  }
  // set the shutdown flag
  shutdown = true;
  // let the threads run on shutdown = true : drop off end and shutdown
  wall->wait();
  // wait for threads to shutdown before destroying wall object
  manager.wait_task(this);
  // remove barrier object
  delete wall;
  wall = NULL;
  // remove mutex;
  delete mutex;
  mutex = NULL;
  // done ..
  return true;
}

int Task_test_manager::Sub_test_manager::svc(void)
{
  // starup of thread .. grab initialization mutex
  mutex->acquire();
  // create our private window queue as part of the
  // Sub_test_manager queue container member creation & local reference building
  win_queues.push_back(vector<Sipwin*>());
  vector<Sipwin*> & queue = win_queues.back();
  queue.reserve(n_elements);
  // done - release mutex
  mutex->release();
  // wait on wall for all threads to finish init
  wall->wait();
  // all threads are done with init ... fall into main loop.
  while (true) {
```

```
    // wait for data
    wall->wait();
    // check for data
    // if no data, time to shut down.
    if (shutdown) {
      return 0;
    }
    // let's work.
    if (!owner.SubProcess(queue)) {
      // error in processing - kill remaining windows and continue
      while (queue.size() != 0) {
      Sipwin* w = queue.back();
      queue.pop_back();
      if (w) {
        delete w;
      }
      }
    }
    // wait for owner to signal that is done.
    wall->wait();
    // continue in loop
  }
}


/*
 *  $Log: task_test_manager.C,v $
 *  Revision 1.75  2000/02/07 15:09:04  shmulik
 *  Fixing print formatting
 *
 *  Code Review by
 *
 *  Revision 1.74  2000/01/19 16:05:49  shmulik
 *  Using now Id_singleton
 *
 *  Code Review by
 *
 *  Revision 1.73  1999/12/23 09:02:02  shmulik
 *  Using int for refer5ence to forward destination insstead of string.
 *  Conversion from input string to int are performed using Id_services
 *  data source.
 *
 *  Code Review by Meir
 *
 *  Revision 1.72  1999/11/29 10:17:46  meirm
 *  Add information on error
 *
 *  Code Review by
 *
 *  Revision 1.71  1999/10/19 17:25:05  shmulik
 *  Adding FORWARD_ALL return status
 *  and replacing old FORWARD return status
 *  by FORWARD_THIS status.
 *
 *  Code Review by
 *
 *  Revision 1.70  1999/09/27 08:51:24  meirm
 *  *** empty log message ***
```

```
*
*  Revision 1.69  1999/09/27 08:46:19  joseph-w
*  removing defects count, changing static object to member
*
*  Revision 1.67  1999/08/29 11:52:51  rina
*  changed due to changes in Unified_event.
*
*  Revision 1.66  1999/08/12 12:57:25  meirm
*  checking subwindows before forwarding to queue
*
*  Revision 1.65  1999/08/12 12:15:24  sharond
*  Adding branch function
*
*  Revision 1.64  1999/06/30 17:22:40  joseph-w
*  KAI KCC 3.4a syntax changes
*
*  Revision 1.63  1999/06/22 17:01:49  shmulik
*  Adding 'IGNORE' return status from function on window.
*  Handling correctly in tasks (test manager, multislice manager, defects
*  handler and in composite function)
*
*  Revision 1.62  1999/06/16 06:07:03  meirm
*  dealing with filtered out defects
*
*  Revision 1.61  1999/06/15 13:17:03  shmulik
*  Adding log message
*
*  Revision 1.60  1999/04/26 06:01:57  shmulik
*  removing WITHDRAW_NON_CONST. Every withdraw will return
*  non constant object.
*
*  Revision 1.59  1999/03/07 07:27:11  joseph-w
*  KCC std c++ complience + dependecies fix
*
*  Revision 1.58  1999/02/28 08:09:21  joseph-w
*  added multithreading support
*
*  Revision 1.56  1999/02/11 14:08:08  sharond
*  *** empty log message ***
*
*  Revision 1.55  1999/02/02 14:30:29  shmulik
*  sdefects bug fix
*
*  Revision 1.54  1999/01/19 15:05:37  shmulik
*  Removing inline from virtual methods
*
*  Revision 1.53  1998/12/29 09:51:04  sharond
*  Get rid of some unneccessarily methods
*
*  Revision 1.52  1998/12/23 12:54:33  shmulik
*  some minor changes
*
*  Revision 1.51  1998/12/22 16:32:06  meirm
*  *** empty log message ***
*
*  Revision 1.50  1998/12/17 15:11:35  shmulik
*  enhancments
*
```

```
*  Revision 1.49  1998/12/09 13:52:25  meirm
*  Remove patch on r_code
*
*  Revision 1.48  1998/12/08 14:00:38  sharond
*  Enable Forwarding of windows to any destination
*
*  Revision 1.47  1998/11/30 13:41:14  shmulik
*  Enhancements in test manager/functions interface (from
*  now function  returns only OK, ERROR or forward)
*
*  Revision 1.46  1998/11/26 12:03:20  sharond
*  Add destination status to windows
*
*  Revision 1.45  1998/11/25 07:03:46  shmulik
*  changing methods returning void into methods returning bool.
*
*  Revision 1.44  1998/11/05 15:52:42  sharond
*  after demo effect
*
*  Revision 1.43  1998/10/27 07:08:19  meirm
*  *** empty log message ***
*
*  Revision 1.42  1998/10/21 09:41:17  joseph-w
*  fixed minor memory leak
*
*  Revision 1.41  1998/10/16 17:22:41  shmulik
*  *** empty log message ***
*
*  Revision 1.40  1998/10/16 11:27:48  shmulik
*  Enhancing the treatement of end of scan condition
*  for queues data sources and tasks of queues.
*
*  Revision 1.39  1998/10/16 07:33:22  sharond
*  *** empty log message ***
*
*  Revision 1.38  1998/10/15 11:06:15  sharond
*  *** empty log message ***
*
*  Revision 1.37  1998/10/14 15:18:04  shmulik
*  updating defect handler so that it will write defect
*  windows.
*
*  Revision 1.36  1998/10/13 06:09:50  joseph-w
*  fixed composite task + istream/ostream/input channel tasks
*
*  Revision 1.35  1998/10/08 07:32:18  sharond
*  *** empty log message ***
*
*  Revision 1.34  1998/10/06 14:04:16  sharond
*  Multi slice Balls handling
*
*  Revision 1.33  1998/09/23 15:00:29  shmulik
*  Adding function element to window's private members
*  and removing it from sipwinpack. making Win_queue
*  to contain pointers to Sipwin instead of pointers to
*  Sipeinpack.
*
*  Revision 1.32  1998/09/17 07:32:41  shmulik
```

```
 *  Improving output of SIP defects
 *
 *  Revision 1.31  1998/09/10 20:02:07  sharond
 *  *** empty log message ***
 *
 *  Revision 1.30  1998/08/31 09:15:44  yaelm
 *  add option to choose type of operation between 2 id-sets in overlap pim
regions, through class Set_op.
 *  use Simple_pim::GetAsPim() instead of direct Pim building.
 *
 *  Revision 1.29  1998/08/25 09:08:20  meirm
 *  *** empty log message ***
 *
 *  Revision 1.28  1998/08/25 07:52:31  sharond
 *  *** empty log message ***
 *
 *  Revision 1.27  1998/07/12 13:50:15  meirm
 *  *** empty log message ***
 *
 *  Revision 1.26  1998/07/09 08:49:36  meirm
 *  *** empty log message ***
 *
 *  Revision 1.25  1998/06/30 14:01:13  meirm
 *  *** empty log message ***
 *
 *  Revision 1.24  1998/06/29 10:04:04  meirm
 *  *** empty log message ***
 *
 *  Revision 1.23  1998/06/10 14:23:53  joseph-w
 *  *** empty log message ***
 *
 *  Revision 1.22  1998/06/10 10:02:45  shmulik
 *  *** empty log message ***
 *
 *  Revision 1.21  1998/06/09 11:25:06  meirm
 *  *** empty log message ***
 *
 *  Revision 1.20  1998/05/26 05:15:41  shmulik
 *  minor fixes
 *
 *  Revision 1.19  1998/05/19 17:41:44  shmulik
 *  Adding data persistance to data sources and the
 *  concept of sync Vs. non_sync data sources used by tasks
 *
 *  Revision 1.18  1998/04/08 07:09:38  eyalk
 *  converting to string, to ansi conventions, new factory, etc ...
 *
 *  Revision 1.17  1998/03/22 08:29:18  meirm
 *  *** empty log message ***
 *
 *  Revision 1.16  1998/03/17 12:47:30  meirm
 *  *** empty log message ***
 *
 *  Revision 1.15  1998/02/19 07:25:00  eyalk
 *  updates of new compiler
 *
 *  Revision 1.14  1998/01/18 19:00:39  shmulik
 *  *** empty log message ***
```

```
*
*  Revision 1.13  1998/01/06 14:11:57  shmulik
*  Replacing all LOG_CONFIG log messages by LOG_APPLIC
*
*  Revision 1.12  1997/12/29 07:46:42  yulia
*  Adding NEW logger
*
*  Revision 1.11  1997/11/04 09:18:00  shmulik
*  *** empty mlog message ***
*
*  Revision 1.10  1997/10/13 14:18:42  shmulik
*  adding support for sipdata_sip_defects
*
*  Revision 1.9  1997/10/12 07:55:10  shmulik
*  *** empty mlog message ***
*
*  Revision 1.8  1997/09/25 06:56:21  shmulik
*  adding general_data variable to func DoSetParams
*
*  Revision 1.7  1997/07/01 11:46:42  shmulik
*  adding support for production of file of defect windows
*
*  Revision 1.6  1997/06/02 08:07:49  shmulik
*  regular updating
*
*  Revision 1.5  1997/05/28 07:32:04  meirm
*  before speed test integration
*
*  Revision 1.4  1997/05/20 17:56:35  shmulik
*  Purification and bug fixes
*
*  Revision 1.3  1997/05/01 11:25:38  shmulik
*  *** empty mlog message ***
*
*  Revision 1.2  1996/11/20 09:18:10  shmulik
*  Improvements in code
*
*  Revision 1.1  1996/10/29 16:25:02  shmulik
*  Initial version
*
*
*/


#endif // MP_SAFE
```